Uso de grafos conceptuales para la visualización de documentos biomédicos y su utilización para la generación de resúmenes

Proyecto de Sistemas Informáticos

Facultad de Informática

Universidad Complutense de Madrid

Departamento de Ingeniería del Software e Inteligencia Artificial

Curso 2013/2014

Raúl Vicente Bueno Sevilla

Marco Gallardo Casu

Richard Jordan Cabana Ramírez

Directores:

Alberto Díaz Esteban

Gonzalo Rubén Méndez Pozo

Raúl Vicente Bueno Sevilla, Marco Gallardo Casu y Richard Jordan Cabana Ramírez, autores del presente documento y del proyecto *Uso de grafos conceptuales para la visualización de documentos biomédicos y su utilización para la generación de resúmenes* autorizan a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

18 de junio de 2014

Raúl Vicente Bueno Sevilla Marco Gallardo Casu Richard Jordan Cabana Ramírez

Queremos agradecer a todas aquellas personas que a lo largo de este díficil pero apasionante año han estado apoyándonos hasta el final del proyecto. Familiares y amigos que nos hacen más llevadero el día a día y nos animan en los momentos más dificiles. A los profesores que gracias a sus esfuerzos y el tiempo que dedicaron a enseñarnos los conocimientos que ahora utilizamos hicieron posible que hoy hayamos podido desarrollar este proyecto. Por último, y no menos importante, a nuestro director Alberto Díaz Esteban y codirector Gonzalo Rubén Méndez Pozo por guiarnos en todo momento y habernos dado la oportunidad de participar en este proyecto tan motivador. A todos ellos gracias.

A mis padres por la paciencia que han tenido durante todos estos años. A Jesús y Álvaro por su gran apoyo durante los momentos más difíciles. A mis amigos de siempre, por su amistad y cariño. A todos los que han creído en mí y me han dado fuerzas para seguir.
Raúl

Resumen

Con la reciente expansión de Internet y la gran cantidad de información a la que podemos acceder, es importante encontrar aquella que sea de calidad y concisa. Gracias a los últimos avances en el procesamiento de lenguaje natural, es posible procesar y transformar en conocimiento dicha información. Sin embargo, todavía se sigue investigando para recuperar la que consideremos de nuestro interés, algo en lo que trabaja la Generación Automática de Resúmenes. Pero no hay que olvidar que representando la información de forma visual somos capaces de comprender y procesar mucho más rápido. Nuestro sistema puede generar resúmenes sobre informes médicos pero intenta ir más allá proporcionando una interfaz gráfica para el tratamiento de dicha información así como su modificación y personalización, intentando facilitar así la comprensión y síntesis de los mismos.

PalabrasClave: Procesamiento del lenguaje natural, recuperación de información, informe médico, detección de conceptos, ontología, *UMLS*, MetaMap, Grafos, Gephi, visualización de información, Generación Automática de Resúmenes.

Abstract

With the recent expansion of the Internet and the large amount of information we can access, it is important to find one that is concise and of high quality. Thanks to recent advances in natural language processing, it is possible to process and transform information into knowledge. However, research is still being done in order to retrieve information that we consider to be of our interest and that is something that the Automatic Text Summarization works on. But we should not forget that by representing information visually, we are able to understand and process it much faster. Our system can generate summaries of medical records, but we try to go beyond that, providing a graphical interface for the processing of such information, as well as its modification and customization, trying to facilitate its understanding and synthesis.

Keywords: Natural language processing, information retrieval, medical report, concepts detection, ontology, *UMLS*, MetaMap, graphs, Gephi, visualization of information, Automatic Text Summarization.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación

Gracias a la rápida evolución de la informática y del nacimiento de Internet, disponemos de una gran fuente de conocimiento a la que podemos tener acceso desde prácticamente cualquier lugar del mundo. La rápida expansión de los teléfonos inteligentes o *smartphones* permiten que se pueda generar y distribuir una gran cantidad de información más rápido que nunca, ya que no es necesario tener un ordenador para ello. Sin embargo, este torrente de información comienza a acarrear un grave problema, el cual está siendo investigado durante los último años, conocido como *infoxication* o sobrecarga de información. Este problema conlleva a que, al haber tanta información, hay que perder más tiempo discriminando la que no nos es útil, llevando asociado una reducción de la productividad, una pobre toma de decisiones e incluso efectos negativos para la salud como se puede ver en numerosos artículos y estudios científicos [6, 15]. Por ello nace el desarrollo de distintas herramientas que ayuden a tratar la información, tanto a profesionales como a particulares, siendo, actualmente, un tema de gran importancia la recuperación, gestión y tratamiento de la información.

En ese ámbito está la cuestión que afrontamos: desarrollar una herramienta que permita apoyar el tratamiento de la información en el campo de la salud. Un tema que lleva siendo objeto de estudio durante varios años, desarrollándose distintas herramientas sobre las que nos apoyamos para conseguir nuestro objetivo. Nuestra propuesta está pensada para poder procesar informes médicos y poder trabajar con la información de una manera visual, siendo capaz de indicar qué conceptos son más importantes o qué grupo de conceptos son más significativos. Con esto acotamos en cierta medida el problema ya presentado de la sobrecarga de información de un simple vistazo.

La generación de resúmenes, no es una tarea fácil, y está siendo objeto de estudio desde hace años para encontrar la mejor forma de tratar la información y así ofrecer lo que realmente el usuario está buscando. Abordamos esta parte para ofrecer resúmenes sencillos sobre la información tratada.

Este proyecto se apoya en otros desarrollados anteriormente. Por una parte el proyecto *Extracción de información de informes médicos* elaborado en el curso 2012-2013 por Irene Sánchez Martínez, Víctor Martínez Simón y Lucía Hervás Martín [10]. Por otra parte, la

tesis *Uso de grafos semánticos en la generación automática de resúmenes y estudio de su aplicación en distintos dominios: biomedicina, periodismo y turismo*, desarrollada por Laura Plaza Morales [16]. Nuestro proyecto toma como partida ideas y trabajo desarrollado para procesar textos biomédicos, así como conocimientos para el procesamiento de grafos y su uso en la generación de resúmenes, ampliando y mejorando este trabajo, así como el desarrollo de un entorno visual para observar cómo se procesa la información.

1.2. Objetivos

Integrando los esfuerzos desarrollados por nuestros compañeros en el pasado, nuestros objetivos se dividían en los siguientes puntos:

- Procesamiento de textos biomédicos para la recuperación de la información.
- Construcción de grafos conceptuales a partir de los textos procesados.
- Visualización de los grafos conceptuales.
- Interacción básica entre el usuario y los grafos conceptuales para la modificación del grafo eliminando nodos.
- Procesamiento de los grafos conceptuales para la visualización de la información más significativa.
- Recuperación de la información procesada en el punto anterior para la generación de resúmenes.
- Generación de resúmenes no sólo usando frases del texto original, para obtener un resumen de mayor calidad.
- Generación de un resumen a partir de varios documentos biomédicos.

Conforme hemos avanzado en el proyecto, nos hemos encontrado con dificultades que nos han obligado a modificar estos objetivos. Por un lado, la investigación y desarrollo a partir del trabajo de Hervás Martín et al. llegó a un punto muerto cuando no pudimos recuperar la información que necesitábamos para la construcción de relaciones. Por otro lado, una lucha continúa contra las limitaciones de algunas herramientas usadas: recuperación de la información, visualización de dicha información, interacción, etc. Estas dificultades nos obligaron a eliminar los dos últimos puntos de los objetivos explicados anteriormente a favor de desarrollar nuevos:

- Mejorar el acceso y la recuperación de la información sobre las relaciones entre conceptos.
- Proporcionar distintas opciones para modificar la distribución visual de nodos y aristas de un grafo.
- Estudio de distintas métricas para hacer agrupaciones de nodos significativos.
- Permitir distintas configuraciones para modificar la apariencia visual de un grafo.

- Aumentar la interacción, resaltando la información de los nodos que se seleccionen, así como la recolocación manual de los mismos.
- Permitir el filtrado por importancia de nodos o grupos significativos de nodos.

1.3. Estructura del documento

El presente documento sigue una división en varios capítulos, los cuáles describimos brevemente a continuación:

Capítulo 1. Introducción: detalla cuál es la motivación de nuestra propuesta y los objetivos que perseguimos con la implementación de este proyecto. Por último anticipa la organización del documento.

Capítulo 2 Estado de la cuestión: en este capítulo se presentan los conocimientos teóricos sobre los que se basa nuestro proyecto a los que se hace alusión en el resto del documento.

Capítulo 3 Arquitectura del sistema: profundizamos en la arquitectura de nuestro sistema y en las herramientas utilizadas, haciendo hincapié en aquellas partes que pueden ser más interesantes o que pueden resultar útiles para alguna de las herramientas usadas.

Capítulo 4 Sistema de visualización de grafos y generación de resúmenes: nos centramos en la funcionalidad del sistema, así como el proceso dividido en etapas desde que un documento biomédico comienza a ser procesado hasta el resultado final de obtener un resumen. También se presenta un caso de estudio.

Capítulo 5 Conclusiones y trabajo futuro: para finalizar, expondremos las conclusiones a las que hemos llegado después de desarrollar este proyecto, así como la funcionalidad que podría ser interesante expandir en un futuro.

Capítulo 2

Estado de la cuestión

2.1. Introducción

Este capítulo surge de la necesidad de explicar los fundamentos más teorícos sobre los que se apoya el proyecto. Además, se presentan las herramientas y bases de conocimiento utilizadas para el objetivo de generar grafos de conceptos y a partir de ellos, resúmenes.

Este capítulo está dividido en las siguientes secciones:

- **Ontologías**: en primer lugar, ya que el objetivo final del proyecto es la de procesar informes médicos y poder sacar un resumen y un grafo asociado a los mismos, es necesario el procesamiento de la información, así como la extracción y representación de la información de una forma que podamos deducir más a partir de ella. Por ello es necesario introducir el concepto de ontología, ya que será un tema recurrente a partir de ahora, además de que usaremos entre otras, SNOMED-CT[1] y UMLS[2]. También se introduciran otros conceptos necesarios y una breve explicación de cómo está estructurada la información que procesaremos.
- **MetaMap**: esta herramienta permite identificar conceptos presentes en un texto. Es necesario ya que introduce técnicas de expansión de acrónimos y desambiguación que nos permitirán identificar los conceptos de forma más exacta.
- **GATE**: GATE[3] es una útil herramienta para el procesado de lenguaje natural o PLN. Nos permitirá procesar textos para que posteriormente *MetaMap* pueda identificar los conceptos descritos en ellos.
- **Generación Automática de Resúmenes**: en esta sección detallaremos cómo se puede representar la información procesada, para poder generar resúmenes. En ella se aclararán algunos conceptos sobre grafos, así como la teoría necesaria en la que se fundamenta el sistema.
- **Visualización y procesado de grafos**: en la última sección del capítulo introduciremos está herramienta, creada para facilitar el trabajar con grafos. Debido a que

[1] SNOMED-CT: *Systematized Nomenclature Of MEDicine - Clinical Terms*.
[2] UMLS: *Unified Medical Language System*.
[3] GATE: *General Architecture for Text Engineering.*

nuestro enfoque es usar grafos para representar la información es necesario contar con una herramienta de este tipo para el procesado de la información.

2.2. Ontologías

Una ontología [22] es un esquema conceptual dentro de uno o varios dominios dados; con la finalidad de facilitar la comunicación y el intercambio de información entre diferentes sistemas y entidades. Nos ayuda a conseguir un mayor nivel de conocimiento sobre un dominio concreto. Es mucho más que un vocabulario o un diccionario de términos. Una ontología suele utilizar diferentes esquemas con estructuras complejas de datos que contienen todos los términos y las relaciones entre ellos. También es normal que contengan atributos que facilitan la búsqueda de información, agrupamiento, indexación...

Lo más importante de la ontología es que captura el conocimiento de un dominio concreto de manera que sea utilizable por sistemas informáticos, por ello la ontología facilita procesos como la búsqueda de información y de apoyo a sistemas de inferencia. Las relaciones de las ontologías suelen ser :

- Jerárquicas : de forma que un término *es un* de otro término.
- Equivalencia : por ejemplo sinonimia, hiperonimia, antonimia, etc.
- Asociativas: para facilitar la recuperación de la información.

Uno de los usos de las ontologías es que sirven de base a varios algoritmos de PLN para extraer información, obtener las relaciones entre los términos, automatización de resúmenes, en definitiva, con obtener conocimiento de textos. También para el etiquetado formal de documentos para reconocer las relaciones entre los términos, mejorar las búsquedas e inferir conocimiento desconocido.

Emplear ontologías que utilizan relaciones de tipo jerárquico entre sus términos como por ejemplo UMLS permiten obtener los descendientes de los términos y obtener nuevo conocimiento de ellos como pueden ser sus sinónimos, antónimos, hiperónimos, etc. Es por ello que cada vez más se están utilizando en campos como la biotecnología o el campo médico y farmacéutico ya que ofrecen mejor respuesta que los tradicionales motores de búsqueda .

Explicaremos algunas de las ontologías utilizadas en este proyecto que nos proporcionan el conocimiento necesario para la generación de los resúmenes SNOMED-CT y UMLS.

2.2.1. SNOMED-CT

SNOMED-CT[4] es la terminología[5] clínica integral, multilingüe y codificada de mayor amplitud, precisión e importancia desarrollada en el mundo.

[4] http://www.nlm.nih.gov/research/umls/Snomed/snomed_main.html

[5] Un conjunto de términos o vocablos de determinada profesión, ciencia o materia. Fuente: RAE: Real Academia Española.

Podríamos decir que una terminología clínica es un conjunto de términos específicos relacionados con el ejercicio práctico de la medicina y fundamentados en la atención de la salud de los pacientes. Tiene un formato multilingüe que proporciona una terminología de referencia para los profesionales de la salud de forma precisa e inequívoca. Se considera un componente de vital importancia para la comunicación segura y eficaz de la información sobre la salud.

Según el Ministerio de Sanidad, Servicios Sociales e Igualdad `SNOMED-CT` contiene más de 310.000 conceptos activos organizados por jerarquías y con definiciones basadas en una lógica formal. Además, contiene más de 728.000 descripciones activas, lo que confiere mayor flexibilidad para la expresión de los conceptos clínicos, y más de 947.000 relaciones definitorias que permiten que la recuperación y análisis de datos sea coherente. La pieza o unidad semántica básica de la terminología son los *conceptos*, estructurados en múltiples jerarquías, que se asocian a descripciones y mantienen relaciones entre ellos. Cada concepto de `SNOMED-CT` tiene un identificador numérico único, permanente y no reutilizable denominado *ConceptID*. La secuencia de dígitos en un *ConceptID* no transmite información alguna relacionada con su significado o naturaleza. El significado de un concepto se representa de manera legible para las personas mediante descripciones. Los conceptos se definen formalmente por sus relaciones con otros conceptos. `SNOMED-CT` presenta los siguientes componentes básicos:

- Conceptos
  - Representan ideas clínicas.
  - Cada concepto tiene un código llamado *ConceptID*.
  - Los conceptos están organizados en jerarquías desde lo más general a lo más específico.
- Descripciones
  - Relacionan los conceptos con descripciones que las personas pueden entender.
  - Un concepto puede tener varias descripciones asociadas, cada una representando un sinónimo que describe la misma idea clínica.
- Relaciones
  - Unen cada concepto a otros que tienen significados parecidos.
  - Estas relaciones proporcionan una definición formal y otras características a los conceptos.
  - Hay relaciones *"es un"* que conectan conceptos en la jerarquía con conceptos más generales. Por ejemplo *virus neumonía* tiene una relación *es un* al concepto más general *neumonía*. Esta relación *es un* define la jerarquía de conceptos de `SNOMED-CT`.
  - Otros tipos de relaciones representan otros aspectos de la definición de un concepto. Por ejemplo el concepto *virus neumonía* tiene una relación *agente causante* al concepto *virus*.

2.2.2. UMLS

En 1986, la NLM[6] inició la construcción del UMLS[7], cuyo objetivo era impulsar el desarrollo de sistemas para la recuperación e integración de información biomédica desde distintas fuentes, incluyendo registros electrónicos de pacientes, bases de datos bibliográficas, bases de datos actuales y otros sistemas expertos. UMLS integra más de 60 familias de vocabularios médicos y biológicos que abordan temas como genes, anatomía, proteínas, enfermedades, en definitiva, una amplia terminología en el campo de las ciencias de la salud.

UMLS presenta tres fuentes de conocimiento: el Metatesauro, el Léxico Especializado y la Red Semántica.

2.2.2.1. El Léxico Especializado

Ha sido desarrollado para proporcionar la información léxica necesaria para el PLN[8]. Se pretende que sea un léxico general de inglés que incluye muchos términos biomédicos. La entrada léxica para cada palabra o término registra la información sintáctica, morfológica, ortográfica, las inflexiones de género y número, las conjugaciones de los verbos, los comparativos y superlativos de los adjetivos y adverbios, e incluso posibles patrones de complementariedad que necesita el sistema de PLN.

2.2.2.2. La Red Semántica

La Red Semántica del UMLS es una de las tres fuentes de conocimiento que se encuentra actualmente en desarrollo en la NLM como parte esencial del proyecto. Por medio de los 132 tipos semánticos, la Red Semántica, garantiza una categorización consistente de todos los conceptos representados en el Metatesauro. Los 53 enlaces entre los tipos semánticos establecen la estructura de la Red y representan las relaciones más importantes en el dominio biomédico.

2.2.2.3. El Metatesauro

Es una ontología biomédica formada por una colección de términos extraídos de diferentes vocabularios controlados y sus relaciones. El Metatesauro UMLS, es el mayor diccionario de sinónimos en el ámbito biomédico, esta fuente proporciona una representación del conocimiento biomédico muy útil para múltiples aplicaciones.

Conceptos e Identificadores de Concepto (CUIs)

Un concepto es un significado, y un significado puede tener diferentes nombres. Cada concepto o significado en el Metatesauro tiene un identificador único y permanente al que nos referiremos como CUI (*Concept Unique Identifier* o Identificador Único de Concepto).

[6] NLM: *National Library of Medicine* (Biblioteca Nacional de Medicina).
[7] *Unified Medical Language System.* Fuente: http://www.nlm.nih.gov/research/umls/
[8] PLN: Procesamiento de Lenguaje Natural.

Términos e Identificadores Léxicos (LUIs)

Los términos agrupan un conjunto de variaciones léxicas de la misma cadena. Por ejemplo, *adenoidectomy*, *ADENOIDECTOMY* y *Adenoidectomies* son diferentes cadenas que hacen referencia al mismo término y reciben en UMLS el identificador único léxico o LUI de L0001425 que a su vez hace referencia al concepto con identificador C0001425.

Nombres de Conceptos e Identificadores de Cadenas (SUI)

Cada nombre de concepto o cadena en cada lenguaje en el Metatesauro tiene un identificador único y permanente o SUI. Cualquier variación en las letras o conjunto de caracteres, de mayúsculas o minúsculas o de un signo de puntuación es una cadena diferente con diferente SUI.

Átomos e Identificadores de Átomos (AUI)

Los átomos son el componente básico de la estructura del *Metathesaurus* y son los nombres de los conceptos o cadenas de cada uno de los vocabularios. Cada vez que una cadena aparece en un vocabulario se le asigna un identificador único o AUI (*Atom Unique Identifier*).

El propósito del Metatesauro es enlazar nombres alternativos y vistas de un mismo concepto, así como identificar relaciones útiles entre diferentes conceptos. En concreto, estas relaciones pueden ser de dos tipos: entre conceptos dentro de un mismo vocabulario o entre con ceptos de diferentes vocabularios. Todas las relaciones se encuentran almacenadas en la tabla mrrel, a excepción de las relaciones de coocurrencia, que se encuentran en mrcoc, y las de equivalencia entre conceptos de distintos vocabularios, que se encuentran en mrmap y mrsmap. Ejemplos de estas relaciones son CHD (*child* o hijo), PAR (*parent* o padre), QB (*qualified by* o "puede ser calificado por"), RQ (*related and possibly synonymous* o "relacionado y posible sinónimo") y RO (*related with* o "relacionado con").

2.2.3. Obsevación

Debido a que es un recurso de usos múltiples que incluye los conceptos y términos de diferentes vocabularios desarrollados para fines muy diferentes, el Metatesauro debe ser personalizado para su uso eficaz en la mayoría de las aplicaciones específicas. Sus decisiones sobre qué incluir en el subconjunto personalizado del Metatesauro tendrán un efecto significativo en su utilidad en sus sistemas. Fuentes de vocabulario que son esenciales para algunos propósitos, por ejemplo, LOINC[9] para el intercambio de datos estándar de laboratorio, pueden ser perjudiciales para los demás, como el PLN. También puede ser importante excluir un subconjunto de los nombres de conceptos que se encuentran en una fuente vocabulario que es otra forma útil, por ejemplo, las abreviaturas no estándar o formas abreviadas que carecen de validez aparente o que producen resultados falsos en el PLN.

[9] LOINC: *Logical Observation Identifiers Names and Codes*.

2.3. MetaMap

MetaMap es un programa desarrollado por la NLM de los Estados Unidos para la traducción de textos biomédicos a conceptos del Metatesauro de UMLS. Cuenta con múltiples opciones de configuración y permite mapear o asociar términos que aparecen en un texto biomédico, con conceptos del Metatesauro UMLS. Utiliza enfoque basado en el PLN y en técnicas lingüísticas. Es una herramienta configurable que nos permite elegir entre varias opciones, entre ellas, si debemos respetar el orden de las palabras del texto, qué terminologías queremos utilizar para identificar los conceptos, como ignorar los términos muy genéricos, etc.

La traducción de los términos de un documento a conceptos del Metatesauro de UMLS presenta diversos problemas, que pueden solucionarse utilizando *MetaMap*:

- Si únicamente se tienen en cuenta términos individuales, en muchos casos los conceptos indexados no reflejarán la verdadera semántica del texto.
- Si se realiza la traducción exacta de una palabra o sintagma nominal, sin considerar posibles variantes léxicas o semánticas, frecuentemente no se recupera ningún concepto, o bien se recuperan conceptos que no son los adecuados.
- Por último, incluso encontrándose el término exacto en el Metatesauro, éste puede ser ambiguo y tener distintos conceptos asociados. Por ejemplo, el Metatesauro contiene dos conceptos para el término *ventilation*, uno relacionado con el flujo de aire en los edificios, y otro relacionado con la respiración. *MetaMap* implementa un algoritmo de desambiguación (opción -y) que favorece a aquellos conceptos semánticamente consistentes con el resto de conceptos en su contexto, a través de la información contenida en sus tipos semánticos [11]. No obstante, la ambigüedad en el Metatesauro sigue siendo, a día de hoy, el talón de aquiles de UMLS.

2.3.1. Funcionalidades

Algunas funcionalidades principales de *MetaMap* son, entre otras, las siguientes:

- **Desambiguación**:
  Uno de los problemas principales que afectan al PLN es la ambigüedad del lenguaje, y una de las mayores debilidades de *MetaMap* es su incapacidad para resolver la ambigüedad del Metatesauro, es decir, las situaciones en la que dos o más conceptos comparten un sinónimo. Es por ello que se incluyó finalmente un sistema de desambiguación o WSD[10] que se puede activar mediante el uso de la opción -y.
- **Detección de negaciones**:
  MetaMap es capaz de hacer uso de una versión extendida del algoritmo *NegEx* para poder determinar cuándo un concepto está negado. Para ello, si se está utilizando el formato de salida por defecto (*human-readable*) es necesario hacer uso de la opción -negex.

[10] WSD: *Word Sense Disambiguation*

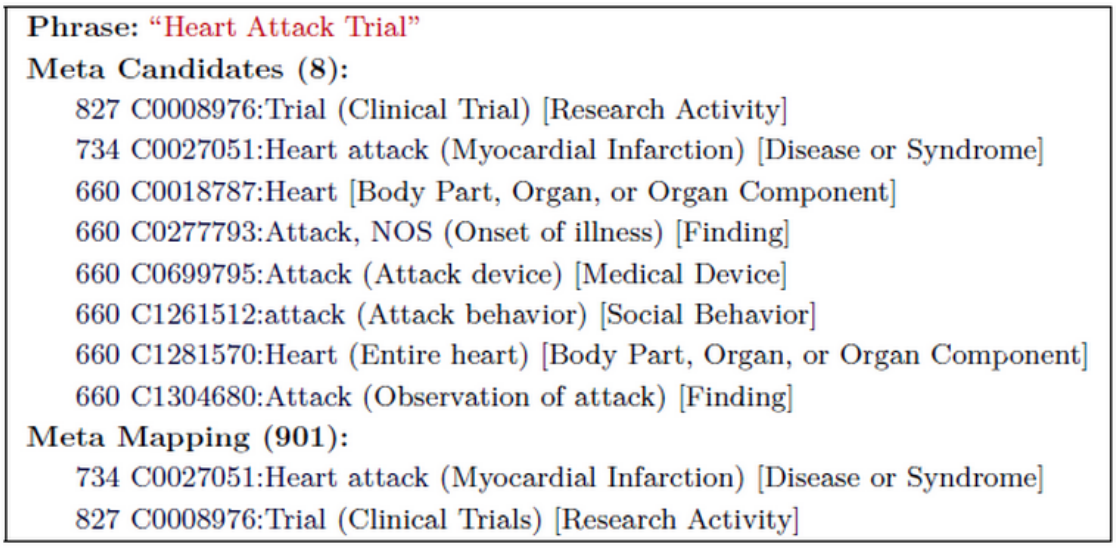


Figura 2.1: Conceptos candidatos para el sintagma *hearth attack trial* y selección de los mejores candidatos.

- **Detección de acrónimos y abreviaturas definidos por el autor**:
  En multitud de textos de dominio técnico aparecen con frecuencia acrónimos y abreviaturas (`AA` del inglés *acronyms and abreviations*) que además suelen ir junto a sus definiciones o extensiones. Se trata de asociar lo que potencialmente es un `AA`, que deberá estar escrito entre paréntesis, con su supuesta expansión, que debe estar situado precediendo al propio `AA` dentro de la misma frase.

- **Mapeo final**:
  Es el paso final del algoritmo de *MetaMap*, en el cuál se construyen los mapeos completos mediante la combinación de los distintos candidatos involucrados en cada una de las partes de la frase. A continuación, se examina cada combinación obtenida y se evalúan de la misma forma que se hace en el caso de un candidato de forma individual, como se expuso en el paso anterior.
  El resultado final del mapeo será el conjunto de candidatos que mejor se ajusten al mapeo de una frase.

2.3.2. Salida Generada

MetaMap puede generar archivos con diferentes formatos de salida:

- ***Human-Readable***: es el formato de salida por defecto y muestra, para cada frase del texto de entrada: la propia frase, una lista de conceptos candidatos del Metatesauro asociados a cada parte de la frase, el mapeo formado al realizar las combinaciones de candidatos asociados a partes disjuntas, y datos complementarios como la puntuación otorgada, u otros opcionales como el `CUI` del concepto (-I), los tipos semánticos (-s), o las fuentes (-G).

- `MMO` ***(MetaMap Matching Output)***: incluye un súper-conjunto de la información de la salida *Human-Readable* y tiene formato de términos *Prolog*. Su utilidad radica en

que así, se permitiría realizar un pos-procesamiento por parte de aplicaciones *Prolog*. Para obtener este tipo de salida, debemos hacer uso de la opción -q.

- XML: debido a la importancia del .xml, también existe la posibilidad de obtener, en este formato, los mismos datos que se presentan en la salida MMO. Esto se puede conseguir mediante las opciones –XMLf –XMLf1 –XMLn –XMLn1. La f indica que el .xml tendrá cierto formato mientras que la n indica que no lo tendrá. El '1' señala que se generará un .xml para un archivo de entrada, mientras que el hecho de no incluirlo, nos quiere decir que obtendremos uno por cada entrada citada.
- ***Colorized MetaMap Output* (*MetaMap* 3D)**: este formato está diseñado para proporcionar de manera visual y mediante colores información para los conceptos mapeados por MetaMap en un texto.
- MMI ***(Fielded MetaMap Indexing Output)***: la opción -f proporciona una salida con múltiples líneas que contienen campos delimitados por tabulaciones. Su contenido es el mismo que el de la salida MMO. Este tipo de salida es utilizado principalmente por el MTI (*Medical Text Indexer*).

2.3.3. Observaciones

Aunque *MetaMap* es una herramienta muy potente en nuestro proyecto hemos utilizado *MetaMap API* cuyas funcionalidades son más reducidas que el programa completo. Al principio pensamos que nos sería suficiente con *MetaMap* para poder construir el árbol semántico y poder generar los resúmenes. Al investigar con más profundidad la API de *MetaMap* nos dimos cuenta que no nos proporcionaba la información suficiente para poder crear el árbol y tuvimos que explorar otras opciones. Cuando estudiamos la tesis vimos opciones de poder crear el árbol utilizando SNOMED-CT y UMLS como base de datos y obteniendo la información directamente de ellos. Finalmente decidimos concentrar nuestros esfuerzos en UMLS para obtener las relaciones y las jerarquías que necesitábamos, no obstante, seguimos utilizando MetaMap para obtener los conceptos de los documentos y comenzar desde ahí el proceso para obtener los resúmenes.

2.4. GATE

General Architecture for Text Engineering o GATE es una suite de herramientas *Java* desarrolladas en la *Universidad de Sheffield*, que comenzó en 1995 y hoy es usada por una amplia comunidad de científicos, compañías, profesores y estudiantes para tareas de PLN de todo tipo, incluyendo Extracción de la información, en varios idiomas.

GATE tiene como objetivo eliminar la necesidad de resolver problemas comunes de ingeniería antes de hacer investigación útil, o reingeniería de procesos antes de convertir los resultados de la investigación en aplicaciones.

Para GATE, todos los elementos que componen un sistema *software* de PLN pueden clasificarse en tres tipos de componentes, denominados resources:

- *Language Resources* (LRs), que representan entidades como documentos, corpora u ontologías.

- *Processing Resources* (PRs), que representan entidades que son, en su mayoría, algoritmos como analizadores, generadores, etc.
- *Visual Resources* (VRs), que representan la visualización y edición de los componentes de la interfaz gráfica.

El conjunto de recursos integrados en GATE recibe el nombre de CREOLE (*Collection of REusable Objects for Language Engineering*). Todos los recursos se encuentran empaquetados en archivos JAR, junto con otros ficheros XML de configuración. Pero además de sus propios componentes, GATE incorpora plugins a otros desarrollados por diferentes organizaciones.

2.5. Generación Automática de Resúmenes

La Generación Automática de Resúmenes o GAR consiste en obtener una versión más reducida de uno o varios documentos mediante el uso de alguna aplicación de ordenador, de manera que el resumen que obtenemos tenga la información más importante del documento o documentos de entrada.

La generación automática de resúmenes es una tarea compleja, ya que hay otras tareas que conlleva implícitamente y que tenemos que manejar:

- **Detección de temas**: se debe delimitar los distintos temas tratados en el texto de entrada.
- **Desambiguación léxica**: hay que resolver la ambigüedad del texto y asociar esos términos con un significado adecuado dependiente del contexto en el que se usan.
- **Resolución de referencias**: se deben resolver las referencias anafóricas y pronominales presentes en el texto.
- **Resolución de acrónimos**: identificar los acrónimos y las abreviaturas y resolverlos para conocer las versiones expandidas de los mismos.

2.5.1. Técnicas

A la hora de clasificar el amplio abanico de técnicas utilizadas en generación automática de resúmenes se pueden adoptar principalmente dos enfoques:

- Distinguir entre técnicas que generan resúmenes mediante extracción y técnicas que generan resúmenes mediante abstracción.
- Distinguir, en función de la profundidad del análisis acometido y del conocimiento empleado entre:
  - Enfoques superficiales.
  - Enfoques basados en la estructura del discurso.
  - Enfoques en profundidad.

Las técnicas de extracción generan resúmenes compuestos íntegramente por material del documento original. Para ello, en una primera etapa o fase de análisis, se limitan a la extracción de segmentos clave del texto; posteriormente, durante la fase de síntesis, se dedican a eliminar la incoherencia y la redundancia, e incluso a resolver referencias anafóricas.

Las técnicas de abstracción generan resúmenes que incluyen contenidos que no están presentes explícitamente en el texto de entrada. Durante la fase de análisis, construyen una representación semántica del texto fuente, mediante la identificación de conceptos genéricos y relaciones entre ellos. La fase de síntesis implica el uso de generación de lenguaje natural para reescribir el texto que conformará el resumen final.

2.5.1.1. Enfoques Superficiales

La mayor parte del trabajo se realiza durante la fase de análisis, si bien dicho análisis continúa siendo bastante superficial. Típicamente, el texto de entrada se escanea, calculando para cada unidad (frase, oración o párrafo) un peso o puntuación indicativa de su importancia. Para ello, se evalúan un conjunto de características para cada unidad, se normalizan y se suman. Durante la fase de síntesis, se extraen las unidades mejor puntuadas y se construye el resumen mediante la simple concatenación de las mismas.

Ahora explicaremos las principales heurísticas que se utilizan para seleccionar las mejores oraciones para los resúmenes.

Frecuencia de palabras

Consiste en buscar las apariciones de las palabras y expresiones que se refieren al tema central del documento. Para cada oración del documento, se anotan las ocurrencias de sus palabras y, utilizando sus respectivas frecuencias, se aplica uno de los muchos métodos existentes para calcular una puntuación para la oración.

Estructura del documento

Obtenemos las palabras clave de los títulos, subtítulos y encabezados ya que en teoría tienen los conceptos más importantes. Utilizando para ello también una lista de parada. Luego se púntuan las oraciones según la presencia de dichas palabras claves.

Localización

La idea principal es que dentro de cada párrafo la primera o primeras oraciones contienen la información más relacionada con el tema tratado en el párrafo. Sin embargo, las posiciones relevantes dependen en gran medida del tipo de documento considerado.

Palabras o expresiones indicadoras

En este caso la idea es que algunas palabras o sintagmas de una oración, aunque no sean en sí mismas palabras clave, aportan ciertas pistas sobre si la oración trata con información relevante.

Uso de aprendizaje automático en la selección de oraciones

La idea es "entrenar" a la máquina para que pueda hacer resúmenes de calidad. Este método se está volviendo muy popular. Para ello necesitamos una serie de documentos con sus respectivos resúmenes hechos a mano de manera que pueda aprendear a dar los pesos adecuados a los distintos atributos.

Ventajas e inconvenientes de los métodos superficiales

Las técnicas estudiadas hasta este punto presentan la ventaja de su relativa sencillez y su bajo coste. Sin embargo, no resultan apropiadas para todos los tipos de resúmenes. Si se trata por ejemplo de resumir textos muy extensos, el ratio de comprensión que se necesita es muy elevado, y resulta imposible de alcanzar sin utilizar cierto grado de abstracción. Además, los resúmenes generados frecuentemente adolecen de falta de cohesión y coherencia.

La cohesión es la unión o conexión existente entre las cosas, es decir, que las oraciones se conectan entre sí mediante procedimientos lingüísticos que permita que sea interpretada con relación a las demás.

La coherencia es una propiedad de los textos bien formados que permite concebirlos como entidades unitarias, de manera que las ideas secundarias aportan información relevante para llegar a la idea principal y comprender el significado global del texto.

2.5.1.2. Enfoques Basados en la Estructura del Discurso

Los enfoques recientes hacen uso cada vez más de un sofisticado análisis del lenguaje natural para identificar el contenido relevante en el documento, y para ello analizan las relaciones entre palabras o la estructura del discurso.

2.5.1.3. Enfoques Basados en Grafos

Investigaciones recientes demuestran que a través de la representación del texto como un grafo, se pueden alcanzar soluciones eficientes para una amplia variedad de tareas, tan diversas como: la desambiguación léxica, la extracción de palabras clave, la categorización de textos, la construcción de tesauros, la recuperación de pasajes, la extracción de información, la generación de resúmenes o la clasificación de sentimientos. En ella, los nodos representan cada una de las unidades textuales en las que se divida el texto, que dependiendo de la aplicación pueden variar desde palabras u oraciones hasta párrafos o incluso documentos. Por su parte, las aristas representan algún tipo de relación entre estas

unidades, relaciones que a su vez pueden ser de naturaleza léxica, sintáctica o semántica. La extracción de oraciones relevantes consiste en identificar las oraciones que actúan como centroides en el grafo. Éste es el tipo de enfoque que usaremos en nuestro proyecto.

2.5.1.4. Enfoques en Profundidad

Las técnicas o enfoques en profundidad generalmente realizan resúmenes mediante abstracción. Abstraer implica realizar inferencias sobre el contenido del texto, e incluso hacer referencia a conceptos previos o a un conocimiento que se presupone. De este modo, es posible conseguir un mayor grado de compresión en el resumen, lo que resulta especialmente interesante a la hora de resumir documentos muy largos o para realizar resúmenes multi-documento.

2.6. Visualización y procesado de grafos

En esta sección introduciremos algunos términos necesarios para el tema que estamos tratando. También comentaremos algunas de las diferentes alternativas que buscamos para este propósito y una breve introducción sobre la herramienta que decidimos utilizar para este fin.

2.6.1. La visualización de la información como grafos

Debido a la naturaleza de los grafos de representarse por un conjunto de nodos y otro de aristas que los relacionan, son una excelente técnica para la visualización de conceptos relacionados de un resumen. Se puede imaginar fácilmente cómo dos conceptos pueden estar relacionados de alguna forma, siendo por ejemplo uno de ellos una enfermedad y otro un tratamiento estableciendo así una importancia implícita sobre los mismos. Esto nos permite representar el texto de un documento en un grafo, pudiendo destacar de una forma totalmente intuitiva y visual aquellas partes que concentran la información más significativa. Sin embargo, esto también puede llevarnos a un problema de comprensión si la representación del grafo no es la más adecuada, ya sea por la distribución de los nodos o por la información que transmiten, haciendo que todo el proceso sea inútil. Por ello una de las necesidades que surgen al trabajar con representaciones de grafos es la de elegir un *layout*. Un *layout* no es más que una estrategia de distribución para colocar los nodos y las aristas de un grafo. Existen distintos tipos de *layouts* [20], de los cuáles presentaremos los tipos con los que trabajaremos en nuestra aplicación por ser los que mejor cubren nuestras necesidades.

2.6.1.1. *Layouts* de fuerza dirigida (*force-based layout*)

Se encargan de modelar el grafo como un sistema físico, usando mecánicas a nivel molecular o mecánicas de muelles. Normalmente estos *layouts* combinan fuerzas de atracción entre vértices adyacentes junto con fuerzas de repulsión entre todos los pares de vértices, buscando que la longitud de las aristas sea pequeña mientras que los nodos estén bien separados, como se puede comprobar en la Figura 2.2.

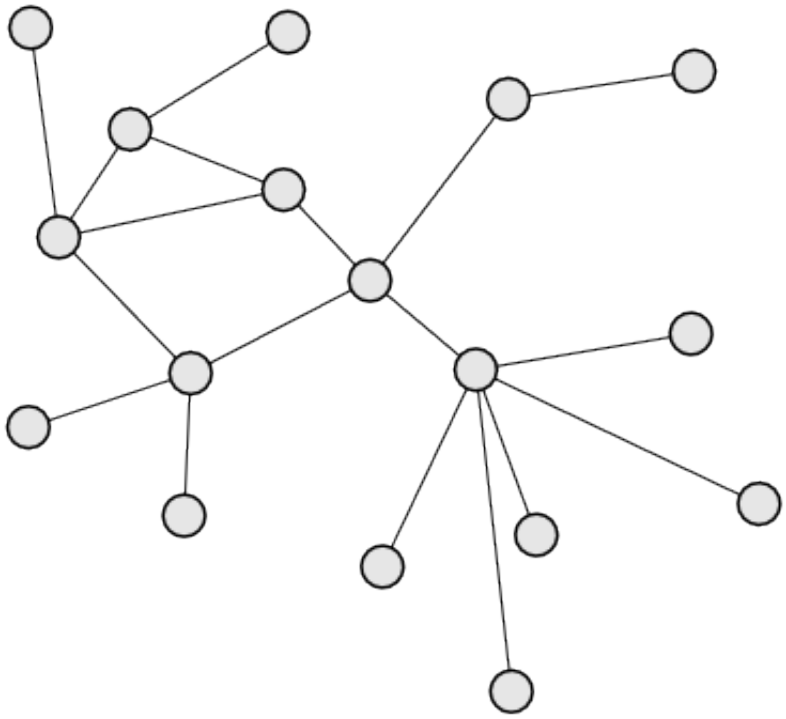


Figura 2.2: *Layout* de fuerza dirigida.

2.6.1.2. *Layouts* por capas (*layered graph drawing*)

Este tipo de *layouts* funcionan mejor para grafos dirigidos sin prácticamente ciclos (`DAG`), como los grafos de dependencias entre módulos o funciones en un sistema *software*. Así, los nodos del grafo se van desplegando en capas horizontales (como si de niveles de un árbol se tratasen), siendo que la mayoría de aristas descenderán a la siguiente capa, tal como se muestra en la Figura 2.3. Existen ciertos algoritmos para evitar el cruzado de líneas o para compactar los grafos y que no se expandan tanto en el espacio horizontal.

2.6.2. Alternativas

Una red libre de escala (*scale-free network*) es un tipo específico de red compleja en la que unos pocos nodos están altamente conectados entre sí (los cuales se denominan nodos *hub*) mientras que los grados de conexión de los nodos restantes es relativamente bajo [1]. Se ha demostrado que los grafos que representan relaciones entre las palabras de textos en inglés forman una red libre de escala [8]. Este tipo de redes también se encuentran en más sitios, como las redes sociales, redes de computadores incluyendo Internet, redes financieras, redes semánticas, entre otras [21]. Esto nos permite encontrar gran cantidad de información con la que poder trabajar con este tipo de redes, además de herramientas diseñadas para trabajar con ellas.

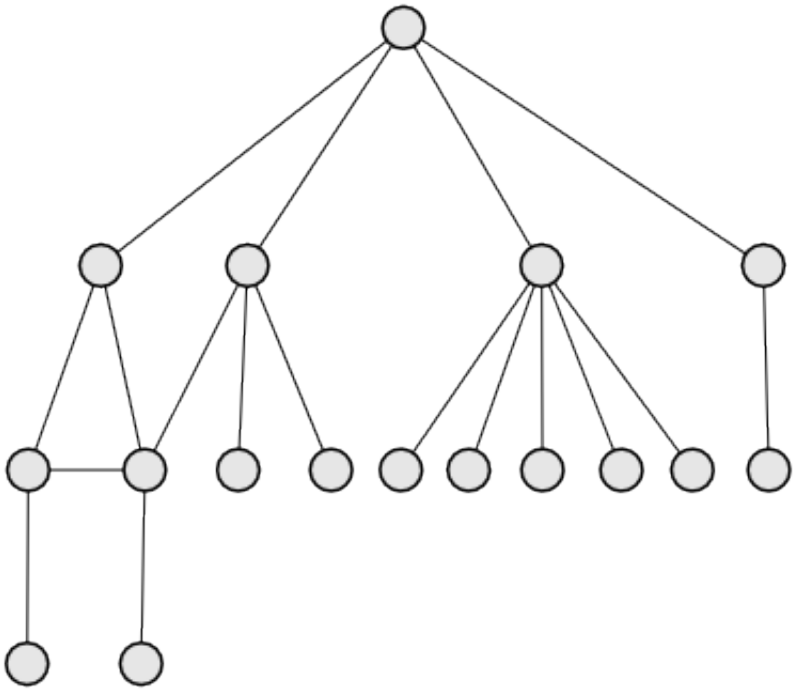


Figura 2.3: *Layout* por capas.

Encontramos varias alternativas de herramientas para la visualización y el procesado de grafos que podrían ajustarse a nuestras necesidades. Las que más nos llamaron la atención fueron *Gephi*[11], *Giraph*[12] y *Graphviz*[13].

2.6.2.1. Giraph

Especializado en grafos especialmente grandes, siendo usado actualmente una versión mejorada en *Facebook*. Tomó ideas de *Pregel*, una arqitectura de procesamiento gráfico desarrollado por *Google* [13]. Con ella puede analizar el grafo social formado por usuarios y relaciones, dando una ligera idea del potencial bruto de esta herramienta. Por contra, es necesario la importación de *scripts* y liberías adicionales, que harían la integración con nuestro proyecto más complicada por tanto quedó descartado.

2.6.2.2. Gephi

No a tan gran escala como *Giraph*, pero puede mostrar varias decenas de miles de nodos sin resentirse según se puede comprobar en su página web. Viene con algunos *layouts* listos para ejecutar y permite integrarse con *Java* a través de un *toolkit* de forma mucho más inmediata que el anterior. Esto nos permitirá poder trabajar con grafos en la herramienta

[11] https://gephi.org/
[12] https://giraph.apache.org/
[13] http://www.graphviz.org/

que desarrollemos, además de poder trabajar con los grafos en la propia aplicación de *Gephi* como un método de depuración adicional. También dispone de algunas métricas y análisis de redes sociales y de redes libres de escala.

2.6.2.3. Graphviz

Graphviz sólo hace la parte de representación de grafos, dando mucha más libertad a la hora de representar el grafo que los anteriores. Por ello, se llegó a presentar como una alternativa solo en la parte de representación de grafos, pero finalmente se descartó para no tener una parte ocupada de la representación de un grafo y por otro lado tener los datos. Por ello elegimos *Gephi*, que mezcla lo suficientemente bien para lo que necesitamos distintas caracteristicas.

Capítulo 3

Arquitectura del sistema

3.1. Introducción

En este capítulo se explicará de forma breve la arquitectura de nuestro sistema así como las clases más importantes de nuestra aplicación. Esto hace posible una mayor comprensión de la aplicación y posibilidad de continuar con el desarrollo de la misma de una forma más sencilla de entender.

Para dar una mejor visión del capítulo se ha dividido en dos partes diferenciadas:

- Objetivos de la arquitectura, restricciones y problemas: se mencionan brevemente las tecnologías usadas así como las librerías de las que hemos hecho uso. Exponemos también cuáles han sido nuestros objetivos a la hora de implementar nuestra aplicación así como las dificultades e imposibilidades técnicas con las que contábamos o nos hemos ido encontrando a lo largo del desarrollo de nuestra aplicación, entrando más en detalle si se considerase necesario en las secciones que correspondan de este mismo capítulo, en la parte de visión lógica.
- Visión lógica: es la sección más importante del capítulo, mostraremos diagramas UML[1] mostrando cómo se compone nuestra aplicación y cómo se relacionan con los otros componentes del sistema. Para dar más claridad a los diagramas, solo mostramos los atributos y métodos no privados y que no sean ni *getters* ni *setters*.

3.2. Objetivos de la arquitectura, restricciones y problemas

3.2.1. Objetivos

Los objetivos de la arquitectura son hacer posibles los objetivos de nuestro proyecto descritos en el capítulo 1. Debido a que hemos partido de un software previo, nos encontrarnos con más problemas técnicos de los que nos imaginábamos, teniendo que abandonar algunos los objetivos como se comenta en dicho capítulo. Los objetivos que tuvimos en

[1] UML: *Unified Modeling Language* (Lenguaje Unificado de Modelado).

mente a la hora de ampliar la funcionalidad y la arquitectura era la de integrar las herramientas que necesitábamos junto con el resto del sistema (*Gephi*) o mejorar la integración de las que ya había (`MMTX`, `GATE`). También se tuvo en cuenta la mejora en eficiencia y calidad de los accesos a la bases de datos para recuperar la información de las relaciones y reducir en la medida de lo posible los tiempos de espera de procesado y construcción de grafos a partir de informes médicos.

3.2.2. Restricciones y problemas

Hemos intentado replicar parte de la funcionalidad desarrollada en la tesis de referencia[16], en el ámbito de los documentos biomédicos. Sin embargo, encontramos numerosos problemas a la hora de implementar dicha funcionalidad (entre otros, versiones de las herramientas usadas y actualizaciones de las bases de datos). Finalmente, conseguimos desarrollar la funcionalidad necesaria para el proyecto.

Las principales librerías que se usa nuestra aplicación son las siguientes:

- `GATE`: la funcionalidad de esta librería se explica en la página siguiente.
- `MMTX`: una librería de una versión algo más antigua de *MetaMap* que se usa en el proyecto de referencia y que hemos seguido usando en nuestra aplicación. *MetaMap* se explica en la página 24.
- `MySQL-Connector-Java 5.1.5`: proporciona los mecanismos para poder conectar con la base de datos y hacer las consultas de `UMLS`.
- `Gephi-Toolkit`: nos proporciona la `API` de *Gephi* para poder operar con los grafos. Ésta, sin embargo, está desarrollada a partir de un entorno de desarrollo específico, generando ciertas restricciones implícitas para ampliar la interacción con el usuario. También tiene ciertas restricciones para pintar los grafos, por tanto los resultados visuales no tienen la misma calidad que usando la herramienta *Gephi*. Puede verse más información en la página 26.

3.2.2.1. Restricciones

El sistema requiere un mínimo de condiciones para que pueda funcionar correctamente. Cuando se distribuye el sistema, se hace con todas las librerías usadas en el sistema, siendo imposible distribuir la base de datos debido a su tamaño.

- *Java Runtime Environment* o *JRE* de *64 bits*, versión 1.7 o superior.
  Para ello desde la página de Oracle recomiendan, para plataformas de 64 bits un procesador mínimo Pentium 2 a 266 MHz. Se necesitará un espacio mínimo de 181MB. En lo referente a la memoria, para plataformas de 64 bits se necesitará una RAM mínima de 128MB para cualquier sistema operativo, se recomienda usar más RAM para aplicaciones que se ejecuten sin un explorador que use el *plugin* de Java. Ejecutarlo con menos memoria de la recomendada puede causar un *swap* en el disco, lo cuál tiene efectos severos en el rendimiento.
  Enlace de descarga: `http://www.java.com/es/download/`.

- *MetaMap*.
  Es un programa desarrollado por la *National Library of Medicine* de los Estados Unidos para la traducción de textos biomédicos a conceptos del Metatesauro de UMLS. Hemos usado la versión 2012 para el desarrollo de la aplicación.
  Enlace de descarga: http://metamap.nlm.nih.gov/#Downloads
  Explicamos cómo se lleva a cabo la correcta instalación de *MetaMap* en el apéndice de Instalación y ejecución de MetaMap y MetaMap Java API, para ello habrá que tener una cuenta de UMLS, se puede solicitar en el siguiente enlace: https://uts.nlm.nih.gov//license.html.

- *Base de datos*.
  Se necesitará tener descargada la base de datos de UMLS para poder realizar las consultas necesarias para la extracción de la información. Como la base de datos de UMLS es muy grande y pesada hemos trabajado con una parte de la base de datos solamente sacada con *MetamorphoSys*. Explicaremos en el siguiente apéndice cómo crear la base de datos (ver Apéndice D).

3.3. GATE

GATE presenta dos modos de funcionamiento. Un modo gráfico y una interfaz para *Java*. El entorno de desarrollo puede utilizarse para visualizar las estructuras de datos producidas y consumidas en el procesamiento, para depurar, obtener medidas de rendimiento, etc.

GATE se distribuye con un sistema de IE llamado ANNIE. ANNIE se basa en algoritmos de estados finitos y el lenguaje JAPE.

3.3.1. ANNIE

Orientado a la extracción de información, ANNIE incorpora un amplio abanico de recursos que acometen tareas de análisis del lenguaje a distintos niveles.

Explicaremos brevemente algunos de los módulos más importantes que lo componen.

- **Restablecer documento**:
  El recurso de reposición del documento permite que el documento pueda restablecer a su estado original mediante la eliminación de todos los conjuntos de anotación y su contenido, además de la que contiene el análisis de formato de documento (original de marcas de revisión).

- ***Tokenizer***:
  El *tokenizer* divide el texto en *tokens* muy simples, como los números, puntuación y palabras de diferentes tipos. El objetivo es limitar el trabajo del tokeniser para maximizar la eficiencia y permitir una mayor flexibilidad al colocar la carga sobre las reglas gramaticales, que son más adaptables.

- ***Gazetteer***:
  Las listas *gazetteer* utilizadas son archivos de texto plano, con una entrada por línea.

Cada lista representa un conjunto de nombres, como los nombres de las ciudades, las organizaciones, los días de la semana, etc.

- ***Sentence Splitter***:
  Es una cascada de transductores de estados finitos que segmenta el texto en frases. Este módulo es necesario para el etiquetador. Utiliza una lista nomenclátor de abreviaturas para ayudar a distinguir puntos y aparte de oraciones-marca de otros tipos.

- ***RegEx Sentence Splitter***:
  Su principal objetivo es hacer frente a algunos problemas de rendimiento identificados en el divisor basado en `JAPE`, sobre todo cuando se enfrentan a la entrada irregular. Se basa en expresiones regulares, utilizando la implementación predeterminada de *Java*.

- ***Part of Speech Tagger:***
  Es una versión modificada del etiquetador *Brill*, que realiza la anotación de cada palabra o símbolo del texto con su categoría morfológica.

- ***Semantic Tagger:***
  El etiquetador semántico de ANNIE se basa en el lenguaje `JAPE`. Contiene normas que actúan sobre las anotaciones asignadas en las fases anteriores, con el fin de producir resultados de las entidades anotadas.

- ***Orthographic Coreference (Orthomatcher):***
  El módulo *Orthomatcher* agrega las relaciones de identidad entre las entidades nombradas encontradas por el etiquetador semántico, a fin de realizar la correferencia.

- ***Pronominal Coreference***:
  Realiza la resolución de la anáfora utilizando la gramática formalismo `JAPE`. Tenga en cuenta que este módulo no se carga automáticamente con el resto de módulos de ANNIE, pero se puede cargar por separado como recursos de procesamiento.

3.4. MetaMap

MetaMap puede utilizarse en un amplio campo de trabajo y para resolver una gran variedad de problemas relacionados con, por ejemplo, recuperación de información, aplicaciones de minería de datos y de texto, clasificación y generación de resúmenes, ampliar o modificar bases de conocimiento. Pero la principal tarea que nos interesa para el proyecto es asociar términos que aparecen en un texto biomédico, con conceptos del Metatesauro `UMLS`.

3.4.1. Conceptos y Algoritmo

La parte esencial de *MetaMap* es aquella que lo define, el mapeo de términos de un texto biomédico a conceptos del Metatesauro `UMLS`. La herramienta utiliza un algoritmo encargado de procesar el texto que consta de cinco etapas distintas, que finalmente devuelve

una salida que esta compuesta por el análisis sintáctico del texto y el conjunto de los posibles candidatos de conceptos UMLS para cada una de las frases del texto. Describimos los pasos más importantes del algoritmo.

3.4.1.1. Análisis del texto en sintagmas

Divide el texto en frases o sintagmas que durante el proceso serán denominadas *phrase*. Dividir el texto en partes simplificará el proceso y se produce una disminución del tiempo de procesado para el documento.

3.4.1.2. Generación de variantes para cada frase

Una variante consiste en una o más palabras de una frase junto con todas sus variantes de habla ortográficas, acrónimos y abreviaturas.

Para ello se utilizan varias fuentes de conocimiento, entre ellas:

- Una base de conocimiento *SPECIALIST* de acrónimos y abreviaturas.
- Una base de conocimiento *SPECIALIST* con reglas de derivación morfológica
- Dos bases de conocimiento de sinónimos.
- El léxico *SPECIALIST*.

3.4.1.3. Para cada una de las frases se recupera el "*candidate set*"

Se forma eligiendo todas las cadenas contenidas en el metatesauro de UMLS que tengan al menos una de las variantes obtenidas de la palabra o frase. Con estos conceptos se devuelve el valor de la función de evaluación .

3.4.1.4. Evaluación de los candidatos

En este punto se calcula el grado de similitud entre una palabra o frase con los candidatos del conjunto obtenido. Se hace una media ponderada de varias propiedades. El resultado está entre 0 y 1.000, donde 0 indica que no existe similitud y 1.000 significa máxima similitud. Las propiedades son :

- **Centralidad**:
  El valor de esta propiedad será '1' si la cadena analizada es la parte principal de la frase, llamada *head* y por el contrario, el valor será '0' si no lo es.
- **Variación**:
  Estima el grado en que difieren las variantes obtenidas de sus correspondientes palabras en la frase original. Para calcular esta variación, en primer lugar se halla el valor de la *variant distance* para cada variante. El valor final de la variación para un candidato, será el promedio de los valores hallados para cada una de las variantes, es decir, si un candidato está formado por la combinación de variantes, el valor final de la variación quedará dividido por el número de éstas.

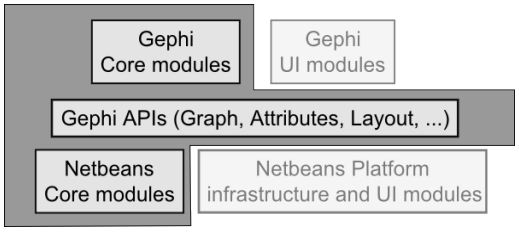


Figura 3.1: Core de Gephi Toolkit respecto a Gephi.

- **Cobertura**:
  Medida calculada a partir del número de términos que participan en el proceso de coincidencia (*match*), tanto de la frase que se está evaluando, como de la cadena obtenida del Metatesauro `UMLS`. Para calcular esta medida, se hallan dos valores, el *Metathesaurus span* y el *phrase span*. El primero es el número de palabras de la cadena devuelta por el Metatesauro que participan en el proceso, y el segundo el número de palabras de la frase original que también participan en él. Finalmente, estos dos valores son divididos, cada uno por la longitud total de la cadena a la que hacen referencia. Para obtener el valor final de la cobertura, se realiza la media ponderada de los dos valores anteriores.
- **Cohesión**:
  Medida similar a la cobertura, pero da especial relevancia a la conexión entre los términos de cada una de las cadenas. Para este cálculo tendremos en cuenta que se denomina como un *connected component*, a la mayor secuencia de palabras contiguas que participan en el mapeo de una determinada cadena. Finalmente, el valor de la cohesión será la media ponderada de la cohesión para la cadena del Metatesauro y del fragmento de texto de entrada que está siendo analizado por *MetaMap*.

Tras obtener el valor de estas propiedades, se realiza la evaluación final de cada candidato, para ello se calcula la media ponderada de las cuatro propiedades vistas anteriormente, dando a la cohesión y a la cobertura un peso doble que a la centralidad y la variación, esta media tiene que ser normalizada para que el valor final esté en el rango de 0 a 1.000.

3.5. Gephi Toolkit API

Es una librería en *Java* desarrollada a partir de *Gephi*, que permite integrar gran parte de la funcionalidad de *Gephi* en nuestros proyectos. Sin embargo, hay algunos módulos que no integra como se puede apreciar en la Figura 3.1 y ciertas características dejan un poco que desear por seguir en desarrollo.

Una de las limitaciones que nos hemos encontrado al usar el Toolkit de Gephi es la escasa interacción que ofrece a nivel visual con el usuario. Otra de las limitaciones, es la forma de expandir cierta funcionalidad: aunque en la comunidad hay varias soluciones alternativas a varias carencias, la mayoría son poco elegantes. Uno de los problemas a los que nos hemos tenido que enfrentar era dotar de cierta interacción del usuario con los grafos. Esto se torna poco trivial, en parte a que *Gephi* esta desarrollado usando el `IDE` *Netbeans*[2] con lo que algunas de estas soluciones no funcionan correctamente en otros

[2]https://netbeans.org/

IDEs, y en parte porque las alternativas ofrecidas pueden añadir a su vez más problemas al usar los módulos de *Netbeans* de forma interna y con poco control para el programador.

3.6. Visión lógica

3.6.1. Visión general

Nuestro programa permite procesar y extraer conceptos significativos de informes médicos usando el proyecto de referencia, así que lo hemos dividido en los siguientes paquetes:

- El Paquete `gephiloaderenum` contiene solamente clases de enumerados que hemos usado durante el desarrollo de nuestro programa.
- El Paquete `gephiloader` contiene las clases principales de nuestra aplicación, toda la interfaz así como la funcionalidad de las partes de *Gephi*.
- El Paquete `graphredef` contiene las clases de los *listeners* que usará *Gephi* para poder interactuar con el grafo.
- El Paquete `main` contiene únicamente la clase principal de nuestra aplicación, por donde entra el programa al empezar la ejecución.
- El Paquete `gps` contiene las clases del GPS[3].

3.6.2. Paquete `gephiloaderenum`

En este paquete están agrupadas las clases de enumerados que facilitan la compresión del código, así como una mejor visión de las opciones de nuestro sistema y su fácil ampliación. A continuación nombraremos las clases y expondremos el uso que tienen:

- `PropertiesNodeLabel`, `ConfigProperties`, `PropertiesNodeGraph` y `Configuration`: Se encargan de las opciones de configuración de los nodos, las etiquetas y las aristas. También de las configuraciones de ejecución de *layouts* y de *clustering*.
- `Heuristics`: Para tener las distintas heurísticas que se aplican en la generación del resumen.
- `Tools`: Define la interacción adicional que permitimos entre el usuario y el grafo.
- `Layouts`: Define los *layouts* que puede aplicar nuestro sistema.
- `Metrics`: Define las métricas que se pueden aplicar en los nodos.
- `MetricsFilter`: Define las opciones de filtrado por importancia de los nodos, en base a su valor.

[3] GPS: *Graph Processing System* (Sistema de Procesamiento de Grafos).

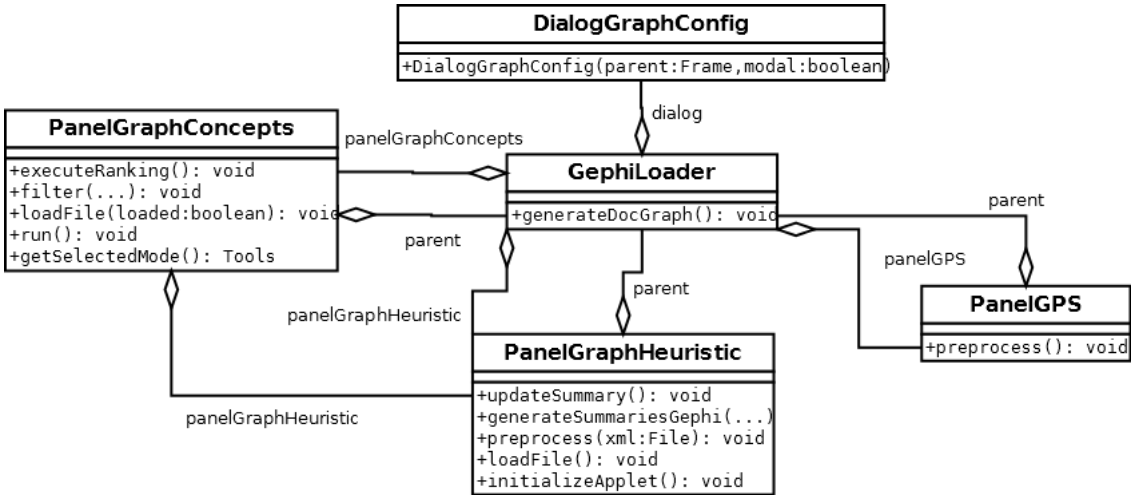


Figura 3.2: Diagrama de clases del paquete `gephiloader`.

3.6.3. Paquete `gephiloader`

Se puede ver el diagrama de clases UML de este paquete en la figura 3.2. También en este paquete existe una clase llamada `ColorComboBoxRenderer`, necesaria para sobreescribir el método *render* de los *ComboBox* proporcionados por *Java* y asi permitir que se muestren las opciones de colores. Esto se usa para seleccionar el color en la parte de filtrado de *clusters*.

3.6.3.1. Clase `GephiLoader`

La clase `GephiLoader` es un *JFrame* cuyo únicos componentes son un *JMenuBar* con los menús de archivo y opciones, y un *JTabbedPane* con tres pestañas que serán los *JPanel* que utilizaremos en nuestra aplicación.

Cada uno de los paneles guarda una referencia a esta clase para poder acceder a los métodos que pueden ser comunes a dos o tres paneles como puede ser crear un documento `.gexf` para el grafo de conceptos en un panel, y el grafo resumen en otro.

A la hora de realizar la aplicación, por si alguien más fuese a coger el código en un futuro para continuarlo le fuese más fácil, hicimos que toda la interacción entre paneles tenga que pasar por este *frame* para no tener que preocuparse porque lo que se necesite esté en una u otra clase, todos los paneles conectan con ésta y ésta conecta con todos.

Esta clase nos permitirá comunicar los tres paneles principales entre sí, así como generar un archivo de configuración que será necesario para la correcta ejecución de la aplicación, donde crea un `.xml` con los diferentes parámetros elegidos para procesar el texto y poder hacer correctamente la extracción de los conceptos del informe que es lo primero que necesitamos obtener.

También tiene un método para crear un solo *JDialog* para las opciones avanzadas y tener una referencia a él para poder consultar las opciones desde el cuadro de diálogo directamente sin tener que almacenar variables intermedias con cada una de las opciones.

Otro de sus métodos genera el `DocumentGraphOBS` con el cuál se construye también el archivo `.gexf` con el grafo para que sea leído por *Gephi*. Este método mapea las frases del documento a conceptos de `UMLS`, saca la jerarquía de ellos haciendo consultas a las tablas `mrconso` y `mrhier` de la base de datos, viendo así los padres de dichos conceptos en `UMLS` dando lugar a un árbol de conceptos.

El archivo con extensión `.gexf` es un archivo que puede leer *Gephi*. En él, se declaran los nodos existentes en el grafo y sus aristas. Pueden tener atributos como tamaño de nodo, color, posición en el lienzo, etc. pero como todo esto será variable en función a las métricas, *clusters* y resto de configuración que usemos para el grafo, nos limitamos a declarar solamente los nodos con su identificador y etiqueta, y las aristas con su identificador, fuente, destino y peso.

La estructura se verá algo como el código 3.1.

Listado de código 3.1: "Estructura de un archivo `.gexf`"

```
1
2  <?xml version="1.0" encoding="UTF-8" ?>
3  <gexf xmlns="http://www.gexf.net/1.2draft" version="1.2">
4          <meta lastmodifieddate="2014-02-26">
5                  <creator>Gephi 0.7</creator>
6          </meta>
7          <graph defaultedgetype="directed" idtype="string" type="
               static">
8                  <attributes class="node" mode="static">
9                          <attribute id="modularity_class" title="
                               Modularity Class" type="integer" />
10                 </attributes>
11                 <nodes count="n">
12                         <node id="0" label="label 0" />
13                         ...
14                         <node id="n" label="label n" />
15                 </nodes>
16                 <edges count="m">
17                         <edge id="0" source="v" target="w" weight="
                               0.5" />
18                         ...
19                         <edge id="m" source="x" target="y" weight="
                               1.67" />
20                 </edges>
21         </graph>
22 </gexf>
```

3.6.3.2. Clase `PanelGPS`

La funcionalidad del `PanelGPS` es simplemente elegir unas pocas opciones para el procesamiento del informe médico a un formato en el que luego podamos generar el grafo, y procesar el informe propiamente dicho. El método *`preprocess`* al principio del procesamiento escribirá las opciones elegidas en el documento de configuración `configOBS.xml`, luego con el archivo del informe usarán los métodos de Laura para el tratamiento de

informes de la clase `OBSDocument`, primero hace un preprocesado del informe, luego se procesa con `GATE` y luego se carga el `.xml` procesado para trabajar con él. Luego se extraen las frases procesadas de este `.xml` y se rellena la tabla con columna de cabecera `DOCUMENT SENTENCES` y se ajusta para que se pueda ver bien el *scroll* horizontal en función de la longitud máxima de las frases procesadas.

3.6.3.3. Clase `PanelGraphConcepts`

Este panel es el que más contenido tiene de nuestra aplicación. En él se mostrará el grafo de conceptos con el que trabajaremos. Tenemos un *JToolBar* con *JButton* y *JToggleButton* para las interacciones con el grafo, tenemos también dos *JPanel* que se ocultan o se muestran según se activen los botones de mostrar las opciones o el filtrado de nodos, y para mostrar la información de los *clusters* y los conceptos en el grafo usamos un *JTextPane*. Aunque lo más importante de esta clase es el *applet* de *Gephi*.

Los métodos más importantes de esta clase son los que ejecutan la configuración del grafo, en este caso, el método *executeRanking* ejecuta el *ranking* para procesar el grafo y adecuarlo a colores y tamaños, también ejecuta la métrica elegida entre los nodos del grafo y prepara los *clusters* de `GPS` para generar el resumen. El método *script* inicializa las variables de las clases `ProjectController`, `PreviewController`, `PApplet`, `Workspace` y `ProcessingTarget` necesarias para el funcionamiento del applet de *Gephi*.

3.6.3.4. Clase `PanelGraphHeuristic`

Esta clase contiene un panel para el *applet* de *Gephi* donde se visualizará el nuevo grafo resumen cuando lo hayamos generado desde la clase `PanelGraphConcepts` y mostrará también en un *JTable* las oraciones que componen el resumen. Al contrario que el grafo de conceptos, el grafo de resumen es sólo para visualizarlo y compararlo con las sentencias extraídas para el resumen, pero no se puede interactuar con él.

3.6.4. Paquete `graphredef`

En este paquete están las clases que sirven para interactuar con el grafo, los *listeners* y las propiedades de los nodos. Se puede ver el diagrama UML en la figura 3.3 en la página siguiente. En ella vemos algunas clases e interfaces que nos proporciona *Gephi*: `PreviewMouseListener`, `NodeRenderer`, `MouseResponsiveRenderer`. *Gephi* usa estas clases internamente para pintar los nodos y asociar las acciones de los *listener* a los nodos cuando éstos son pintados.

3.6.4.1. Añadir funcionalidad de *listeners*

Ha sido necesario extender la funcionalidad de algunas clases que tenía *Gephi* para poder añadir cierta interacción con los nodos, ya que esta funcionalidad no estaba pensada inicialmente cuando se desarrolló. Como hemos presentado anteriormente, se trata de extender la funcionalidad base de cómo se renderiza el grafo para añadir un *listener* que nos pueda comunicar en qué posición de la pantalla se ha pulsado. La clase que finalmente

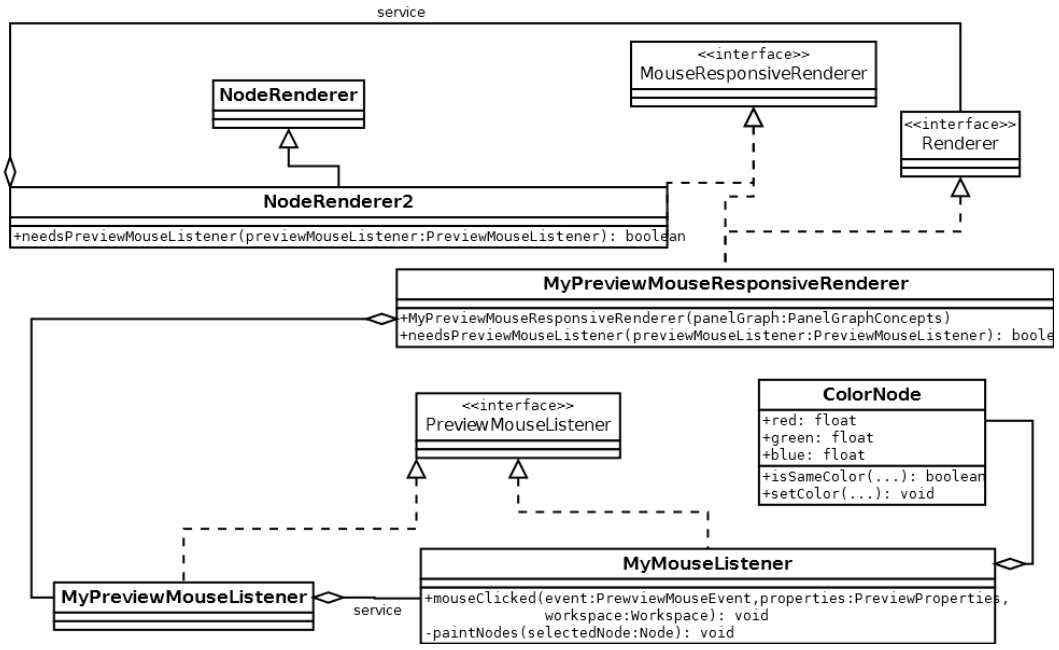


Figura 3.3: Diagrama de clases del paquete `graphredef`.

hace las acciones de los *listener* es `MyMouseListener`. Uno de los grandes problemas de hacer los *listener* de esta forma en la que perdemos parte de control, es que nos fuerza a usar de algún modo algún atributo estático de otras clases para permitir saber que nodo se ha seleccionado; otro problema añadido es el de no poder tener varios paneles que permitiesen una mínima interacción con el grafo al estar compartido el atributo de manera estática.

Para llevar a cabo esto es necesario escribir nuestra propia clase que implemente las clases `MouseResponsiveRenderer` y `Renderer` de *Gephi*, que se encargan de que se puedan captar respuestas del ratón cuando se renderiza la imagen del grafo. En esta clase es exactamente donde se asocia nuestro *listener* del tipo adecuado a la parte que se encarga de renderizar, justo como el método que se muestra en el código 3.3. Además, tendremos que asociar a esta clase el servicio del *renderer* , añadiendo el código 3.2 antes de la definición de la clase (precisamente por esto no funciona correctamente en otros `IDE` como *Eclipse*.

Listado de código 3.2: "Modificación del render"

```
1
2 @ServiceProvider(service = Renderer.class)
```

Listado de código 3.3: "Modificación para usar nuestro listener"

```
1
2 public boolean needsPreviewMouseListener(PreviewMouseListener
      previewMouseListener) {
3         return previewMouseListener instanceof
              MyPreviewMouseListener;
4 }
```

Por último, es necesario crear una clase del tipo `PreviewMouseListener`, sobreescribiendo el correspondiente servicio (3.4) y que incluya la funcionalidad real que queremos que tenga cuando se pulse con el ratón en pantalla 3.5.

Listado de código 3.4: "Modificación del listener del ratón"

```
1
2 @ServiceProvider(service = PreviewMouseListener.class)
```

Listado de código 3.5: "Modificación para usar nuestro método al hacer *click* con el ratón"

```
1
2 public class MyMouseListener implements PreviewMouseListener {
3         public void mouseClicked(PreviewMouseEvent event,
              PreviewProperties properties, Workspace workspace) {...}
4 }
```

En ese caso modificamos el método *`mouseClicked`*, que es el que se llamará en cuanto se haga un click en cualquier parte del área de la pantalla (que es importante aclarar que la y se contempla de forma inversa), permitiendo obtener las correspondientes coordenadas y comprobar si se encuenta algo que nos interese remarcar ahí, usando un radio alrededor del punto en dónde se ha pulsado.

3.6.5. Paquete `main`

Este paquete solo contiene la clase `MainClass` que establece el *`LookAndFeel`* apropiado al sistema operativo y crea una instancia de `GephiLoader`.

3.6.6. Paquete `gps`

Contiene parte del desarrollo del proyecto del curso anterior. Como algunas anotaciones rápidas, en ella hay clases para encargase del procesado de los textos (GateProcessing), de la representación del documento procesado (`DocumentGraphOBS`, `GraphOBS` y `SentenceGraphOBS`) y también de *clustering* y extracción de frases para el resumen (`SFGCOBS` y `SentenceExtractionOBS`).

Capítulo 4

Sistema de visualización de grafos y generación de resúmenes

4.1. Introducción

Este capítulo está dedicado a explicar el proceso que sigue un documento para transformarlo a un grafo de conceptos equivalente al que poder aplicarle distintas transformaciones para por último obtener un resumen basado en esa información. Para ello presentaremos las etapas que sigue este proceso:

- Preprocesamiento: paso previo a la generación del resumen por el cual el informe sigue un tratamiento para filtrar información poco significativa: se discriminan términos muy generales y se eliminan partes del documento poco relevantes. Posteriormente, el documento se divide y se agrupa en distintas oraciones que se usarán en el resto de etapas.
- Generación de grafos de conceptos: en esta etapa se procesan las oraciones y se extraen los conceptos médicos asociados. Posteriormente se construye un grafo procesando esos conceptos y buscando sus relaciones en la base de datos. Por último este grafo se guarda con la interfaz de entrada de grafos de *Gephi*.
- Visualización de grafos: esta etapa ofrece una visualización del grafo generado en el paso anterior. Además, ofrece opciones para modificar la apariencia general del grafo, modificar la distribución de sus nodos (*layout*), interacción para el movimiento o eliminación de algunos nodos y por último, el filtrado de nodos. Esta etapa y la siguiente están altamente conectadas por la necesidad de procesar el grafo para obtener información y la de tratarlo a nivel visual para representar la información de la forma más significativa posible.
- Generación de *clusters*: en esta etapa se introducen técnicas de procesamiento de grafos para aplicar métricas sobre los nodos y poder determinar qué nodos son los más importantes. También se recoge parte de dicha información para hacer agrupamientos (*clusters*) de esos nodos.
- Generación de resúmenes: en la última etapa se asignan las oraciones del documento a los *clusters* generados anteriormente. Con ello se lee puede asociar métricas a las

oraciones para determinar su relevancia. Finalmente, a través de varias alternativas que presentaremos en la sección, se seleccionan aquellas frases más relevantes para generar un resumen simple.

- Caso de estudio: Para terminar el capítulo, mostraremos un caso de estudio de un texto real, en el que se explicará el proceso desde que se procesa un texto hasta que obtenemos un resumen.

Como apoyo a las siguientes explicaciones de cada una de las secciones, se puede consultar la Figura 4.1, en la que se da una visión global de todo el sistema.

4.2. Preprocesamiento

Antes de generar el resumen hay que tratar el documento del informe de manera especial para prepararlo para las próximas etapas. En esta fase se realizan las siguientes acciones:

- Se eliminan aquellas partes que no influyen en la generación del resumen. Como dependen del dominio y del tipo concreto del documento, el sistema permite clasificarlas como "irrelevantes" a la hora de generar el resumen en archivo `.xml` de configuración.
- Se eliminan aquellos términos que por ser muy generales, no nos servirán a la hora de diferenciar frases relevantes o no para el resumen. Entre estos términos se encontrarían las preposiciones, las conjunciones, los artículos, los determinantes, etc. Para esto se utiliza una *stop list* o lista de parada que se elaborará considerando las particularidades del dominio del trabajo.
- En el caso de que el documento incluya un título y un *abstract* o resumen por el propio autor, se extrae el contenido de ambas secciones, separándolas del *body* o cuerpo del documento.
- Finalmente, el *body* se divide en frases. Para ello se utiliza la librería `GATE` (3.3), invocando a componentes de la librería `ANNIE` desde el `API` correspondiente. Seguidamente describiremos el proceso realizado:
  - Iniciar `GATE` y cargar el *plugin* de `ANNIE`.
  - Crea un corpus que encapsula los documentos que se desean procesar.
  - Establecer los módulos que se utilizarán al procesar el documento en el orden que habrán de ser aplicados, estos módulos son `DefaultTokenizer`, `DefaultGazetteer` y `SentenceSplitter`.
  - Ejecutar los módulos anteriores sobre el corpus generado.
  - Al ejecutarlo se creará un `.xml` por cada uno de los textos del corpus, donde se identificarán las distintas frases que lo forman.

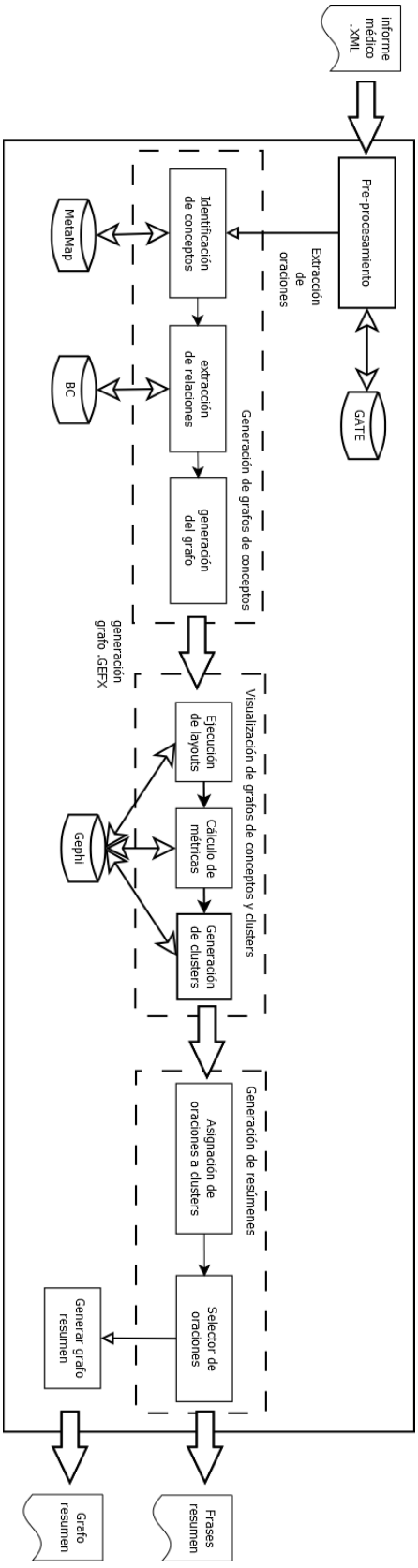


Figura 4.1: Funcionamiento del sistema.

4.3. Generación de grafos de conceptos

4.3.1. Extracción de conceptos

En esta etapa hay que traducir el léxico del informe a conceptos de UMLS. Para ello se usan mecanismos de desambiguación para distinguir entre los diferentes significados del concepto en el contexto en que se encuentre.

Cada sentencia del texto de entrada sera analizada por el algoritmo de desambiguación WSD[1] y se obtiene de la base de datos una lista de conceptos. Puede suceder que haya términos en la oración sin concepto identificado en la base de datos o que varios términos diferentes sean relacionados con el mismo concepto en UMLS.

Finalmente, se puede realizar un segundo filtro para eliminar aquellos términos que se puedan considerar muy generales, y que por tanto no aportan ninguna información para el resumen.

Leyendo un archivo llamado GPS.xml[2] crearemos otro de configuración que hemos denominado configGPS.xml El arhivo de configuración tendrá apartados de categorías con diferentes propiedades, así tenemos las categorías DOCUMENT_PREPROCESS, TAG, STOPLIST (la lista de parada que se comentó en la sección anterior), ONTOLOGY, UMLS, UMLS_DB y SNFC.

De aquí, los que tendremos que modificar a la hora de usar la aplicación por primera vez y al crear la base de datos serán:

- UMLS: tiene la propiedad IGNORED_SEMANTIC_TYPES que es una lista separada por comas con los conceptos generales que no queremos que salgan en nuestro grafo, se pueden añadir o quitar más según se prefiera.
- UMLS_DB: esta categoría tiene cuatro propiedades de configuración para la correcta conexión a la base de datos:
  - DB_NAME: el nombre de la base de datos, en nuestro caso *umls*.
  - URL: el host donde se haya la base de datos, si está en local como en nuestro caso sera *//localhost/umls*.
  - USER: nombre de usuario de la base de datos.
  - PW: contraseña del usuario de la base de datos.

4.3.2. Extracción de relaciones y Generación del grafo

En esta etapa se construye un grafo por cada oración del documento, de manera que tenga su estructura semántica y las relaciones entre sus términos.

Para ello, los conceptos extraídos de cada frase en la anterior etapa se expanden con los conceptos de los niveles superiores en la jerarquía de la base de datos (conceptos más generales de los que descienden los términos del informe). Por lo tanto, se necesita que la

[1] WSD: *Word sense disambiguation*. Algoritmo usado por MetaMap (3.4)
[2] GPS: *Graph Processing System.*

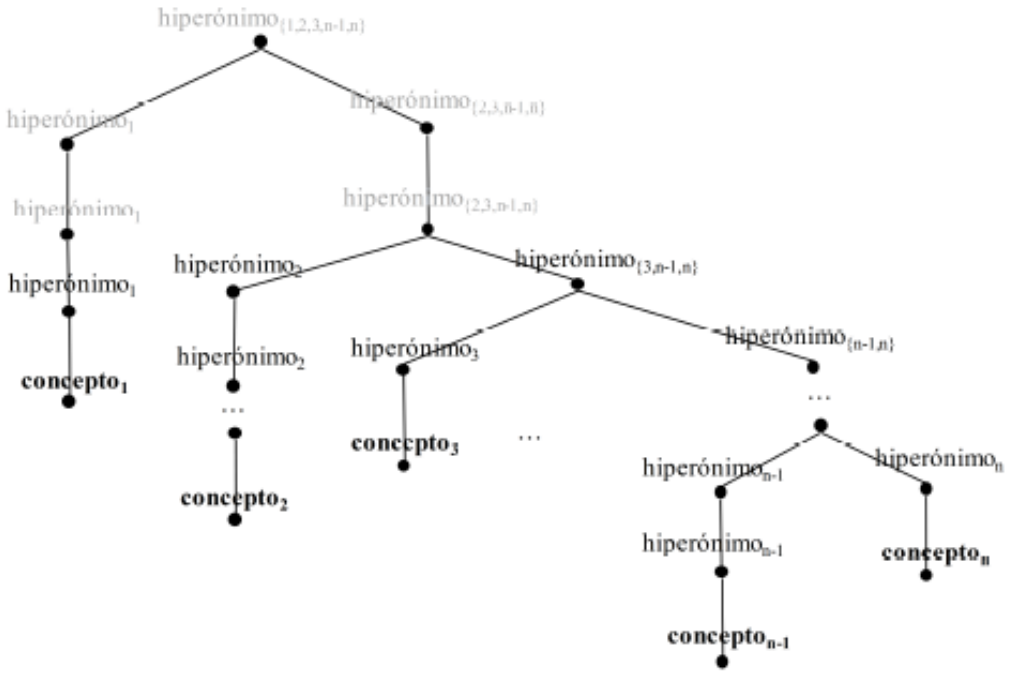


Figura 4.2: Grafo semántico de una sentencia.

```
The goal of the trial was to assess cardiovascular mortality and morbidity for
stroke, coronary heart disease and congestive heart failure, as an evidence-based
guide for clinicians who treat hypertension. While event rates for fatal
cardiovascular disease were similar, there was a disturbing tendency for stroke
and a definite trend for heart failure to occur more often in the doxazosin
group, than in the group taking chlorthalidone.
```

Tabla 4.1: Texto biomédico para procesar.

base de datos de donde se extraigan los conceptos, contemple la relación de hiperonimia entre ellos.

Posteriormente, después de expandir todos los conceptos con sus hiperónimos, las distintas jerarquías se combinan en un único grafo que representa a la oración. En ese grafo, cada vértice es un concepto diferente, no hay conceptos repetidos aunque se repitan en la oración y las aristas representan relaciones semánticas entre esos conceptos.

Por ultimo, los conceptos que se encuentran arriba de la jerarquía representan una información muy genérica por lo que se elimina del grafo los *n* niveles superiores. Nosotros eliminamos el más alto de todos que suele corresponder al concepto genérico de `SNOMEDCT`. En la Figura 4.2 se muestra el aspecto que presentaría el grafo de una frase, mostrándose de un color más claro los conceptos ignorados por ser muy genéricos.

En la Figura 4.3, se muestra el aspecto que presenta el grafo de conceptos del texto presentado en el cuadro 4.1.

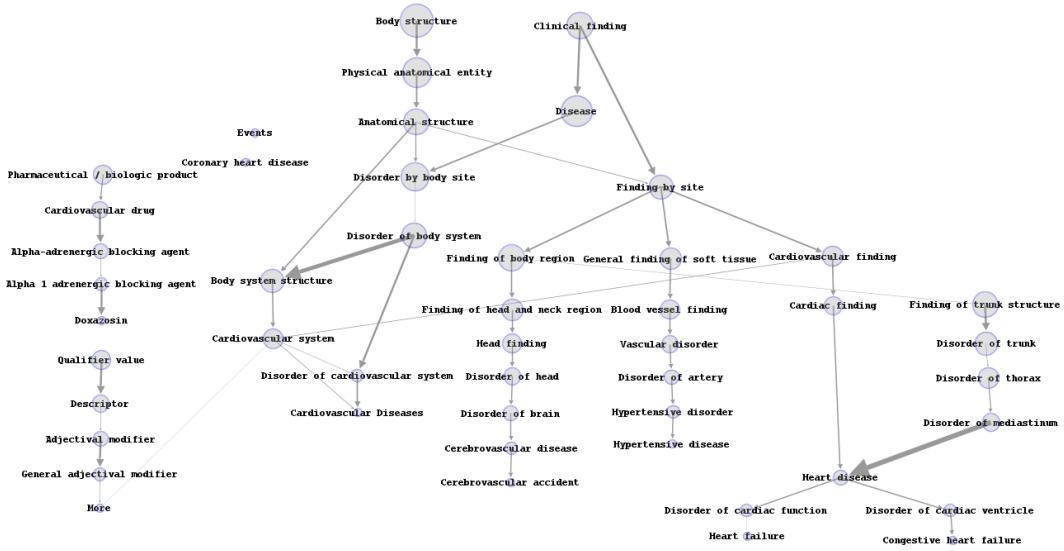


Figura 4.3: Grafo semántico correspondiente al texto de ejemplo.

4.4. Generación de *clusters*

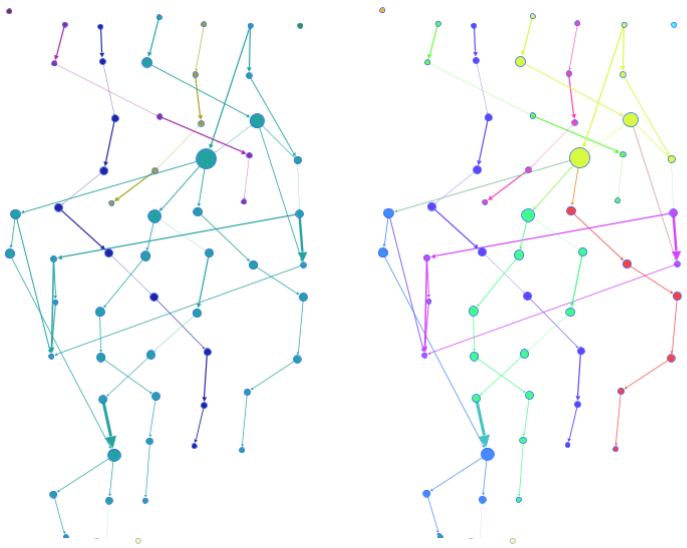
Como hemos comentado en el capítulo 2, estamos trabajando con redes libres de escala (*scale free networks*). Éstas tienen unos poco nodos altamente conectados, conocidos como nodos *hub,* y serán los que tengamos que identificar en primera instancia a la hora de hacer los *clusters*. Dependiendo de la forma de hacer los *clusters,* o de la métrica escogida para asignar una valoración numérica a la importancia de un nodo, éstos pueden variar bastante y dar lugar a distintos resúmenes. A continuación se explican las dos alternativas que sigue nuestro sistema para generar los *clusters*.

4.4.1. Generar los *clusters* usando Gephi

Gephi incorpora un algoritmo de detección de comunidades por el cual permite hacer *clusters* con una configuración limitada: podemos hacer que el algoritmo fuerce a que haya menos o más *clusters* a través de una medida umbral (denominada *resolution*). Sin embargo no podemos establecerla para todos los grafos, ya que dependiendo del grafo o del número de conexiones del mismo esa medida umbral deberá modificarse para obtener mejores *clusters*.

Gephi usa el algoritmo detección de comunidades de Louvain (*Louvain community detection algorithm*) [3], el cual sirve para identificar comunidades en grandes redes. Consiste en dos fases:

- Buscar pequeñas comunidades optimizando la modularidad (*modularity*) local.
- Agregar los nodos de una misma comunidad, creando una nueva red en la que esos nodos son las comunidades.
- Estos pasos se repiten de forma iterativa hasta que se alcanza un máximo de modularidad.



(a) Alta *resolution* (b) Baja *resolution*

Figura 4.4: Efectos del atributo *resolution* en la generación de *clusters* por Gephi.

Como hemos presentado, este algoritmo se basa en algo llamado modularidad. La modularidad [14, 12] es una medida de la estructura de grafos que valora la fuerza de división que tiene una red en distintos grupos o comunidades. Una alta modularidad indica que hay conexiones sólidas entre los nodos de un mismo módulo, pero escasas conexiones entre nodos de distintos módulos (por ejemplo, las redes biológicas presentan un alto grado de modularidad). La optimización de la modularidad (que usa el algoritmo de detección de comunidades de Louvain) tiene un límite de resolución [9] debido a que su algoritmo sigue un modelo nulo global, que asume implícitamente que cada nodo puede quedar unido a cualquier otro de la red (suposición razonable en redes muy grandes), pudiendo llegar a una situación en que la modularidad interprete que dos grupos tienen una fuerte correlación y que se fusionen ambos grupos. Como comentábamos al principio, Gephi nos permite variar este límite, logrando en gran medida paliar sus efectos, pero obligándonos a hacer un análisis previo de la red para establecer de alguna forma cuál es el mejor valor para evitar tener muy escasos *clusters* que no aporten ninguna información o tener demasiados que tampoco sirvan de nada. Podemos ver los efectos de modificar el límite de resolución a la hora de generar los *clusters* en la figura 4.4.

Al usar este método para hacer los *clusters*, hemos podido comprobar que se generan más que usando otras técnicas. Esto puede ser un problema si se generan demasiados de tal forma que no se pueda asociar de la mejor manera las oraciones a los *clusters*. Como se ha comentado, esto se puede regular modificando la resolución, pero tampoco es una panacea: se podrá forzar a que haya menos agrupamientos, a costa de tener uno más grande que contenga prácticamente todos los nodos que guardan conexión con el mismo, mientras que los nodos que no tienen conexiones, pertenecerán a otros agrupamientos distintos. Una buena práctica que hemos seguido al realizar las pruebas, es relajar el valor de la resolución a valores cercanos a 1.0 cuando los grafos son extensos y bien conectados. Si los grafos tienden a ser pequeños y poco conectados, es mejor aumentar el valor entre 2 y 5 puntos para no obtener demasiadas comunidades.

4.4.2. Métricas para la valoración de nodos

Antes de poder generar los *clusters* usando las valoraciones de cada nodo individual es necesario presentar las distintas métricas que usamos para valorar la importancia de los nodos. Cada una de ellas refleja la importancia de un nodo desde distintos enfoques. Presentaremos aquí las cuatro métricas que hemos usado para valorar los nodos [5, 17].

4.4.2.1. *Salience*

También denominada prestigio [7], es la suma de todos los pesos de todas las aristas que tienen como origen o destino a dicho vértice, más en concreto como dictamina la ecuación 4.1.

$$salience(v_i) = \sum_{\forall e_j \| \exists v_k \wedge e_j conecta(v_i, v_k)} peso(e_j) \tag{4.1}$$

4.4.2.2. *Betwenness Centrality*

Denominada también intermediación [18], cuantifica el número de veces que un nodo actúa como un puente a lo largo del camino más corto entre otros dos nodos, por tanto mide la importancia de un nodo como conector de otros nodos. Queda definida por la ecuación 4.2, en dónde σ_{st} es el número total de los caminos más cortos entre los nodos s y t, mientras que $\sigma_{st}(v)$ es el número de esos caminos que pasan a través de v.

$$betwenness(v) = \sum_{s \neq v \neq t} {}^{\sigma_{st}(v)}/_{\sigma_{st}} \tag{4.2}$$

4.4.2.3. *Closeness Centrality*

Denominada también cercanía [18], calcula la suma o bien el promedio de las distancias más cortas desde un nodo hacia todos los demás. Se puede interpretar como la accesibilidad de un nodo en la red. Esto nos permite identificar los nodos más accesibles de la red, dejando más de lado las ramas poco conectadas. Queda definida por la ecuación 4.3.

$$closeness(v) = \sum_{t \in V \setminus v} 2^{-d_G(v,t)} \tag{4.3}$$

4.4.2.4. *Degree Centrality*

Denominada centralidad de grado [18], es la más simple de las medidas de centralidad. Corresponde al número de enlaces que posee un nodo con los demás, siendo la más fácil de calcular. Nos da una información directa acerca de que el nodo más conectado es el más importante, aunque puede no ser cierto si los pesos de las aristas que llegan a él son extremadamente pequeños, por ello la métrica *Salience* es más adecuada que ésta. Se

puede definir como la ecuación 4.4, en la que se calcula el grado de un nodo j como la suma de la existencia de aristas (siendo $a_{ij} = 1$ si existe la arista (i, j) y $a_{ij} = 0$ en caso contrario) desde ese nodo hasta los demás.

$$degree(j) = \sum_{i} a_{ij} \tag{4.4}$$

4.4.3. Generar los *clusters* usando métricas de nodos

Siguiendo la estela del proyecto en el que nos basamos, hemos decidido rescatar esta característica y mejorarla. Así pues, en el anterior proyecto sólo había una métrica que se calculaba en cada nodo para saber su importancia y a través de la que se formaban los agrupamientos. Hemos ampliado esta posibilidad a usar una de las cuatro métricas descritas en la sección 4.4.2.

El algoritmo que se usa en este caso es uno similar al propuesto por Erkan and Radev, que hemos tenido que adaptar para la integración con Gephi y la recuperación de la información para visualizarla. Se pueden encontrar más detalles en el capítulo 3.

El proceso se describe en los siguientes pasos, una vez calculada la valoración de cada nodo usando una de las métricas de la sección 4.4.2:

- Al tratarse de una red libre de escala, se buscan los nodos más conectados en primer lugar, obteniendo n nodos concentradores o nodos *hub* (que definimos a priori dependiendo de las características del grafo y del resumen), los cuales representan los nodos más importantes del grafo.
- Se agrupan los nodos concentradores formando conjunto de vértices concentradores o `HVS` (*Hub Vertex Sets*), que constituyen los centroides[3] de los *clusters* a construir.
- Se comprueba para cada par de `HVS` su conectividad interna para fusionarlos, ya que la conectividad entre los conceptos dentro de un *clusters* ha de ser máxima, mientras que entre distintos *clusters* ha de ser mínima (de forma parecida al algoritmo de detección de comunidades de Gephi), definiéndose por tanto medidas de intra-conectividad y inter-conectividad (ecuaciones 4.5 y 4.6) para asignar estos valores.

$$intra - conectividad(HVS_i) = \sum_{\forall e_j | \exists v,w \in HVS_i \wedge e_j conecta(v,w)} peso(e_j) \tag{4.5}$$

$$inter - conectividad(HVS_i, HVS_j) = \sum_{\forall e_j | \exists v \in HVS_i, \exists w \in HVS_j \wedge e_k conecta(v,w)} peso(e_k) \tag{4.6}$$

- Por último, se asignan los nodos que aún no pertenecen a ningún grupo (los que no están en ningún `HVS`). Para ello se comprueba la conectividad que presentan con cada *cluster*, calculando para ello el grado de conexión según la ecuación 4.7, reajustando los `HVS` y los nodos asignados en un proceso iterativo.

[3] El centroide de un *cluster* se define como el punto equidistante de los objetos pertenecientes a dicho *cluster*.

$$conectividad(v, HVS_i) = \sum_{\forall e_j | \exists w \in HVS_i \wedge e_j conecta(v,w)} peso(e_j) \quad (4.7)$$

4.5. Visualización de grafos

Una de las partes más importantes de nuestro proyecto es la de visualización de grafos de conceptos, por ello es fundamental que la información que representan quede muy clara y se pueda trabajar fácilmente con ella. En esta sección presentaremos todas las posibilidades para trabajar con el grafo y obtener resultados de forma visual, tanto de *clusters* como de métricas aplicadas a nodos. También presentaremos las opciones de filtrado y de interacción con el grafo.

4.5.1. Interacción con el grafo

A pesar de que la API de Gephi aún está muy poco desarrollada en el ámbito de la interacción con el usuario (al contrario que lo desarrollado en la aplicación de Gephi), hemos ampliado la funcionalidad básica de moverse por el espacio del grafo a:

- Selección de nodos: el sistema permite seleccionar nodos, de uno en uno, que cambiará de color para poder identificarlo. También permite resaltarlo en la lista de nodos.
- Movimiento de nodos: permite seleccionar un nodo y después elegir una nueva ubicación a la que se cambiará. Útil para reorganizar pequeñas partes del grafo que pudiesen quedar muy congestionadas.
- Borrar nodos: permite eliminar nodos del grafo, eliminando automáticamente las conexiones que tuviese. Con ello se pueden eliminar nodos que pensemos que no aporten nada o ver cómo se modifica un resumen eliminando ciertos nodos.

4.5.2. Visualización de nodos y de *clusters*

Se han proporcionado algunas configuraciones que permiten aumentar la legibilidad del grafo así como la identificación de los nodos más importantes:

- Tamaño de nodos: El tamaño de los nodos es proporcional a la importancia de dicho nodo (calculada con una de las métricas ya expuestas). Esto permite una identificación visual rápida de los más importantes (ver la en la figura 4.5). Además, se permite variar dicho tamaño máximo y mínimo para intensificar o atenuar esta característica.
- Visualizado de *clusters*: Cuando se generan los *clusters* se colorean usando un color aleatorio, al igual que hace Gephi cuando le toca detectar las agrupaciones. Esto favorece que se identifiquen los *clusters* al momento. Además, para mejorar la visualización, hemos proporcionado una lista de todos los nodos, agrupados por *cluster* y que al seleccionar uno de ellos en el grafo se marcan en dicha lista.

- Apariencia: Hemos incluido opciones para alterar la apariencia del grafo en general y mejorar la legibilidad:

  - Modificación de la opacidad de nodos y aristas.
  - Modificar el tamaño de las etiquetas de los nodos o hacer que dependan del tamaño del nodo, cambiar el color u ocultarlas.
  - Modificar el tamaño de las aristas y de la flecha, además de ocultarlas.

4.5.3. *Layout* del grafo

Presentados en el capítulo 2, detallamos en esta sección los distintos *layouts* que se pueden usar para alterar la distribución de los elementos del grafo [4]. Al contrario de lo que se pueda pensar, un *layout* no se ejecuta inmediatamente: es necesario que pase un tiempo, que pueden ser desde unos pocos segundos a unos minutos, para que puedan recolocar toda la información. Éste tiempo de ejecución depende en gran medida del tamaño del grafo, pero también depende del *layout* que se desea aplicar. En nuestro caso los grafos no llegan a tener una dimensión considerable (en comparación con una red social por ejemplo), por lo que los tiempos de espera serán muy reducidos aunque comentaremos en cada uno de ellos cuánto es el tiempo recomendado.

También hay que aclarar que estas distribuciones permiten ciertas configuraciones para tener cierto control a la hora de distribuir los nodos, pero en ningún caso podremos colocar exactamente como planeamos los nodos debido a su propia forma de colocarlos.

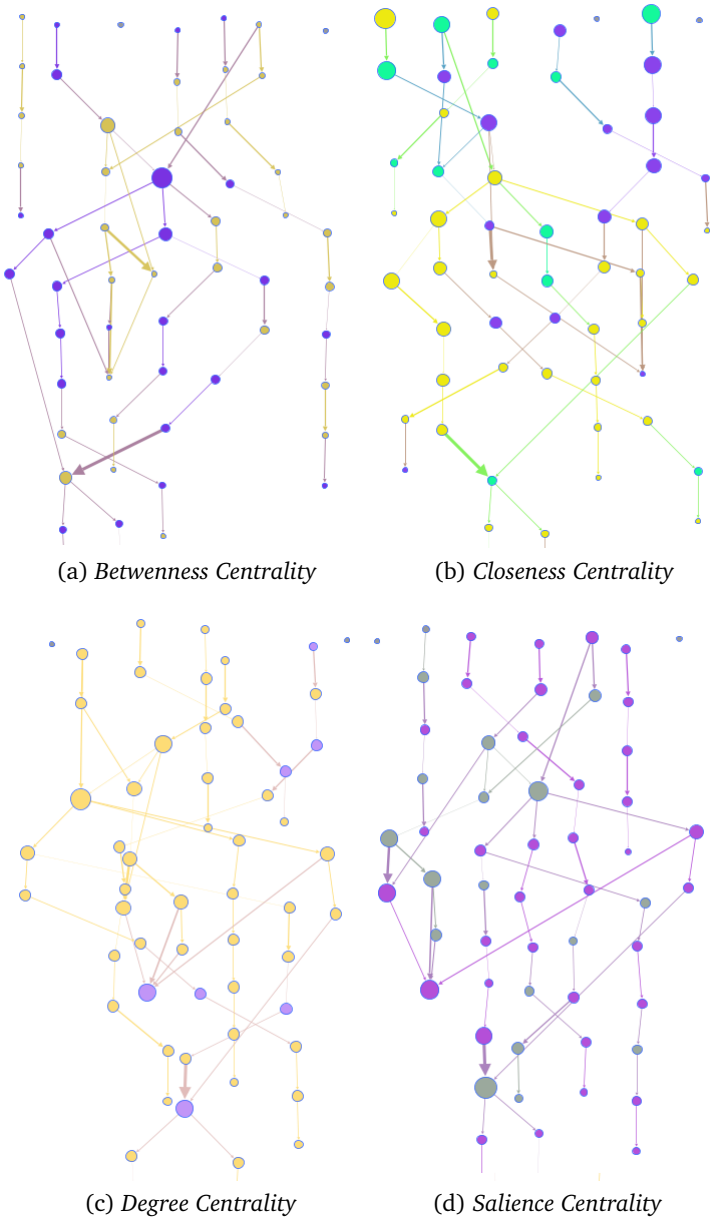
Gephi permite la ejecución de varios *layouts* seguidos en un tiempo especificado (para grandes redes en donde se desee combinar varios para obtener una mejor distribución), pero hemos decidido limitar ésta característica a dos *layouts* como máximo para focalizarnos en el resultado, poniendo a disposición del usuario varias opciones de configuración.

A continuación explicaremos los *layouts* utilizados, aunque se pueden añadir más al proyecto, consiguiendo los *plugins* desde el *market* de Gephi[4] y transformándolos a librerías de Java en ficheros `*.jar`. En el apéndice E se puede encontrar más información.

4.5.3.1. Yifan Hu *Layout*

Este tipo de esquema de distribución pertenece a los denominados de fuerza dirigida. Este *layout* en concreto, aplica técnicas de algoritmo multinivel, pudiendo trabajar con grandes cantidades de nodos (siendo su rango óptimo entre 100 y 100.000) aplicando para ello poco tiempo. A pesar de no usar los pesos de las aristas, aproxima las fuerzas de repulsión haciendo *clusters* usando una simulación de Barnes-Hut [2, 19]. Como podemos apreciar en la figura 4.6, este algoritmo distribuye los nodos por *clusters*. Este *layout* es el más rápido del que disponemos y no necesita más de unos pocos segundos en completar la distribución.

[4] https://marketplace.gephi.org



(a) *Betwenness Centrality* (b) *Closeness Centrality*

(c) *Degree Centrality* (d) *Salience Centrality*

Figura 4.5: Variación de la importancia de los nodos en base a cada métrica y su efecto en la formación de *clusters*.

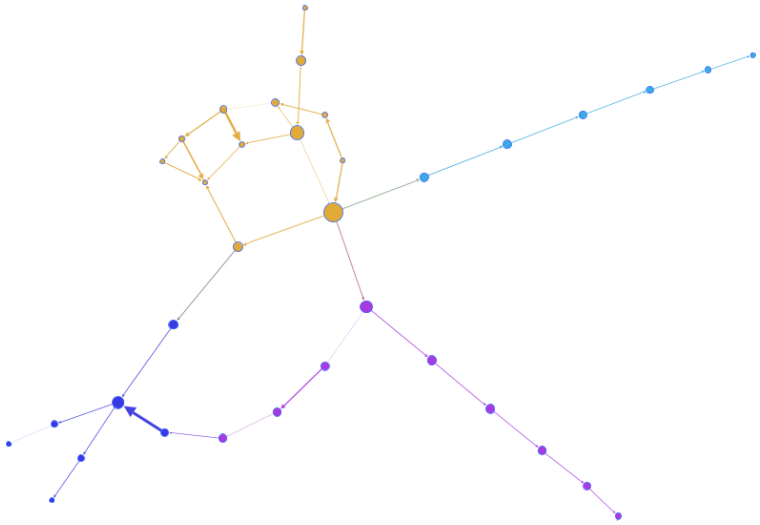


Figura 4.6: Yifan Hu Layout. (Sólo se muestra una porción del grafo)

4.5.3.2. *Force Atlas Layout*

Otro esquema de fuerza dirigida, especializado en redes libres de escala como los grafos con los que trabajamos. A priori debería ser la opción más acertada, ya que a diferencia del Yifan Hu, deja mayor separación entre los nodos que pertenecen a un mismo *cluster*, lo que favorece una mejor lectura de las etiquetas de los mismos. Sin embargo, está peor optimizado, requiriendo más tiempo que el anterior (se recomienda dejarlo ejecutando más segundos que el de Yifan Hu) y con una capacidad de menos nodos.

Siempre se puede ejecutar este *layout* como complemento a alguno de los demás para forzar que los nodos se separen y mejorar la visibilidad general.

4.5.3.3. DAG Layout

Este esquema es de capas, de tal forma que se especializa en mostrar los grafos sin ciclos dirigidos en forma de árbol, como se puede visualizar en la figura 4.8. Las siglas vienen de *Direct Acyclic Graphs*. Muy útil para comprobar los ancestros de los nodos. Por contra, su configuración está muy limitada ya que la separación entre nodos está prefijada por el propio algoritmo, pudiendo hacer que las etiquetas de los nodos se superpongan si el *Label Adjust Layout* no está activo. También intenta que las aristas resultantes no sean muy largas para obtener un grafo más compacto, lo que propicia que haya cruces entre nodos y haciendo que sea menos legible. Es uno de los *layouts* que necesita más tiempo para completarse, sin embargo, según las pruebas realizadas, a veces es mejor dedicarle bastante menos tiempo del que necesita para obtener árboles mucho más legibles.

4.5.3.4. Noverlap Layout y Label Adjust Layout

Comentamos brevemente estos esquemas incluidos, cuya funcionalidad únicamente mejoran la visualización final del grafo. *Noverlap* se centrará en que los nodos no se solapen,

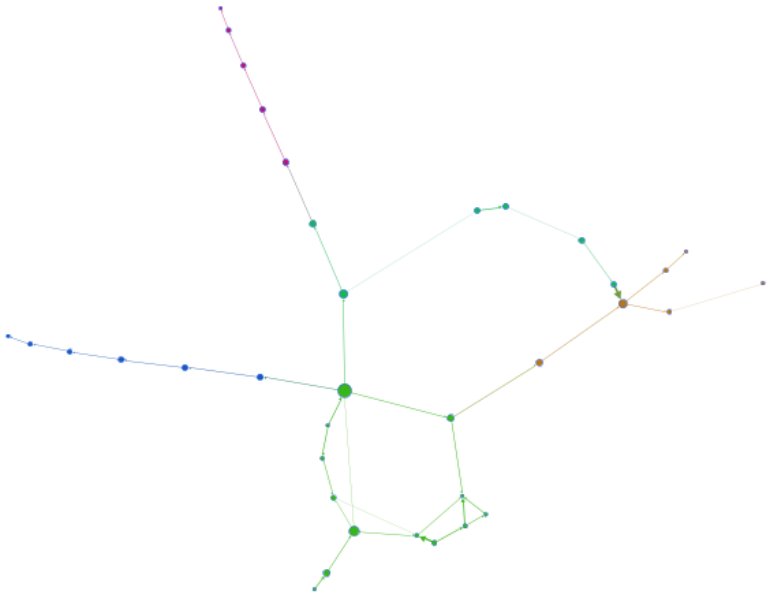


Figura 4.7: Force Atlas Layout

Figura 4.8: DAG Layout

mientras que *Label Adjust* se centrará en que las etiquetas no se solapen. Existe una opción para que cuando se hayan ejecutado los *layouts* con el tiempo asignado, se ejecute a continuación el *layout* para ajustar las etiquetas, el cual parará en cuanto no pueda o no queden más etiquetas por descubrir. Éste último se ejecuta siempre posteriormente a los anteriores si está marcada la opción.

4.5.4. Filtrado de conceptos

Por último, al trabajar con grafos más extensos, nos dimos cuenta de que era necesario habilitar alguna forma de filtrar la información, ya que eran grafos tan grandes que provocaban el problema de *infoxication* que justo queremos solucionar. Para ello hemos dejado dos opciones de filtrado (sólo se puede aplicar una al mismo tiempo) que nos han parecido útiles en nuestras pruebas.

4.5.4.1. Filtrado por importancia de conceptos

A partir de la valoración máxima de un nodo, dividimos los nodos en cinco grupos, determinando a qué agrupación le corresponde cada nodo en base a su valoración. Estos grupos van desde los menos importantes (generalmente los nodos con muy escasas conexiones) hasta los más importantes (los nodos que contienen la mayor cantidad de conexiones). También actualizamos la tabla de información para que se pueda ver rápidamente a qué *clusters* pertenecen dichos nodos.

4.5.4.2. Filtrado por *clusters*

Se puede seleccionar el color del *cluster* que se quiere visualizar, de tal forma que sólo se verán los nodos de dicho color y *cluster*. Además, actualizamos la información de la tabla para que sólo aparezcan en ella los nodos correspondientes.

4.6. Generación de resúmenes

4.6.1. Asignación oraciones a *clusters*

Cuando ya hemos agrupado todos los conceptos, hay que asignar estos grupos a cada una de las oraciones. Necesitamos por tanto definir una medida de similitud entre el *cluster* y el grafo de la oración. Como ambas representaciones son distintas en cuanto a tamaño se refiere, métricas clásicas de similitud de grafos no resultan adecuadas. Por tanto, usamos un sistema de votos, como el que se había desarrollado en el proyecto anterior, basado en la idea de Yoo et al. [23], pero modificando cómo se realiza la votación. Debido a la inclusión de Gephi y de su capacidad de generar los *clusters*, no podemos usar la misma medida que se usaba en el proyecto anterior, el cual tenía en cuenta si un nodo estaba en el *cluster* y además en el `HVS` del mismo. Como ahora no podemos contar con la información de los `HVS` actualizados si Gephi es el que genera los *clusters,* simplificamos la medida de similitud, de tal forma que nos sirva independientemente de la forma por la generemos los *clusters*. Para ello, usaremos la valoración del nodo, haciendo que el sistema de votación se

oriente a que si un vértice está en un *cluster*, éste suma su valoración a la similitud con el mismo, dando mucho más peso a que un concepto extremadamente importante aparezca en una frase a que lo hagan varios sin importancia. Esto queda definido en la ecuación 4.8. En la que $w_{kj} = valor(v_k)$ en caso de que exista en el *cluster*, y 0 en otro caso. ($valor$ dependerá de la métrica seleccionada en el apartado 4.4.2).

$$similitud(O_j, C_i) = \sum_{v_k | v_k \in O_j} w_{k,j} \tag{4.8}$$

4.6.2. Selección de oraciones para el resumen

El último paso de todo el proceso, determina qué N oraciones del texto original son las que formarán el resumen (que vendrá determinado por el ratio de compresión, a 30 % por defecto), en función del sistema de votación descrito en el anterior apartado. En el sistema de partida nos encontramos con tres propuestas para seleccionar las oraciones más importantes que son las que formarán el resumen final que detallamos a continuación:

- Heurística 1: Parte de la hipótesis de que el *cluster* de mayor tamaño es que el que contiene el tema principal del documento, por tanto es el único que se tiene en cuenta para la generación del resumen, seleccionando las N oraciones con las que presenta mayor similitud. Así, queremos que el resumen sólo tenga información del tema principal.

- Heurística 2: Se parte de un enfoque distinto a la anterior heurística, manteniendo la idea de que todos los *clusters* contribuyen a la construcción del resumen con un número de oraciones proporcional a su tamaño. Se seleccionarán de cada *cluster* un número de oraciones que tengan la mejor similitud, dependiendo del tamaño del mismo.

- Heurística 3: Es la más compleja de todas, ya que calcula una puntuación para cada frase, haciendo la suma de las puntuaciones para cada frase, promediado con el tamaño del *cluster,* siguiendo la ecuación 4.9.

$$puntuacion(O_j) = \sum_{C_i} \frac{similitud(C_i, O_j)}{\mid C_i \mid} \tag{4.9}$$

En la siguiente sección expondremos un caso de estudio sobre estas métricas aplicadas sobre un ejemplo real y detallaremos cual de ellas nos ha parecido la que mejor se ajusta a los resultados esperados.

4.7. Caso de estudio

Como apoyo a este capítulo, desarrollaremos el proceso de generación de un resumen usando el texto a resumir del cuadro 4.2. Es un texto lo suficiente extenso como para poder cubrir todos los aspectos desarrollados en los anteriores puntos.

```
The goal of the trial was to assess cardiovascular mortality and morbidity for
stroke, coronary heart disease and congestive heart failure, as an evidence-based
guide for clinicians who treat hypertension. While event rates for fatal
cardiovascular disease were similar, there was a disturbing tendency for stroke
and a definite trend for heart failure to occur more often in the doxazosin
group, than in the group taking chlorthalidone. The results of ALLHAT regarding
doxazosin were first made public by a presentation at the meeting of the American
College of Cardiology, March 2000, and the subsequent publication. Several news
agencies published reports of the study, and comments appeared in a few medical
journals. There was, however, no action taken by either the United States Food
and Drug Administration (FDA) or the Joint National Committee on Prevention,
Detection, Evaluation, and Treatment of High Blood Pressure, which authored the
most recent advisory guideline for the National Heart Lung and Blood Institute
[3]. In May 2001, however, after initiation of a class action lawsuit and a
Citizens Petition, the FDA held a hearing on the issue. Sidney Wolfe and the
author gave presentations recommending that all physicians receive a warning with
an interpretation of the ALLHAT results, and that the labeling and indications
for doxazosin be changed by Pfizer, the company which developed and markets
doxazosin as Cardura. Pfizer's representatives argued that doxazosin is a safe
and effective antihypertensive drug, based on their own accumulated studies, with
no need for any additional warning or change in labeling. The remainder of this
article will summarize the basis for a warning and address the arguments of those
who conclude that nothing further needs to be done.
```

Tabla 4.2: Documento de ejemplo procesado

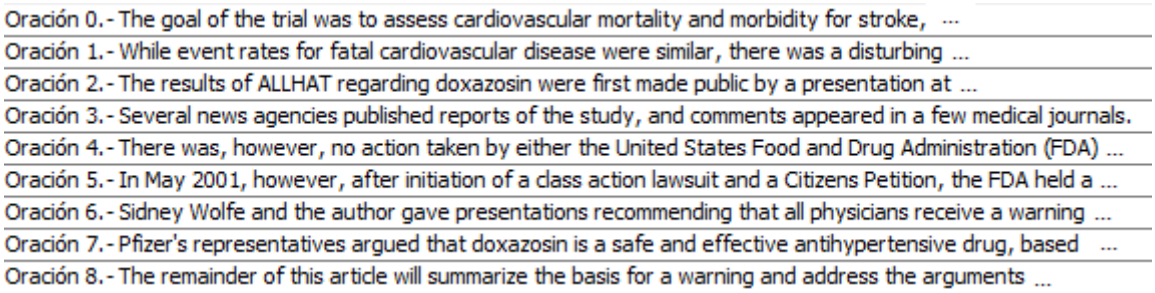


Figura 4.9: Lista de oraciones extraídas del documento 4.2.

4.7.1. Generación de grafos de conceptos

En primer lugar, procesamos el documento, con la configuración por defecto para eliminar aquellos conceptos que no aportan nada gracias al pre-procesamiento. Con ello obtenemos en total 9 oraciones que se pueden consultar en la figura 4.9. Posteriormente se procede a la identificación de conceptos y la construcción del grafo de conceptos que podemos comprobar en la figura 4.10 (se han ocultado las etiquetas para visualizar mejor el grafo).

4.7.2. Generación de *clusters* y visualización del grafo

El siguiente paso es elegir un *layout* y una forma de generar los *clusters*. En nuestro caso elegiremos el Yifan Hu *Layout* (4.5.3), y elegiremos que los *clusters* se generen basados

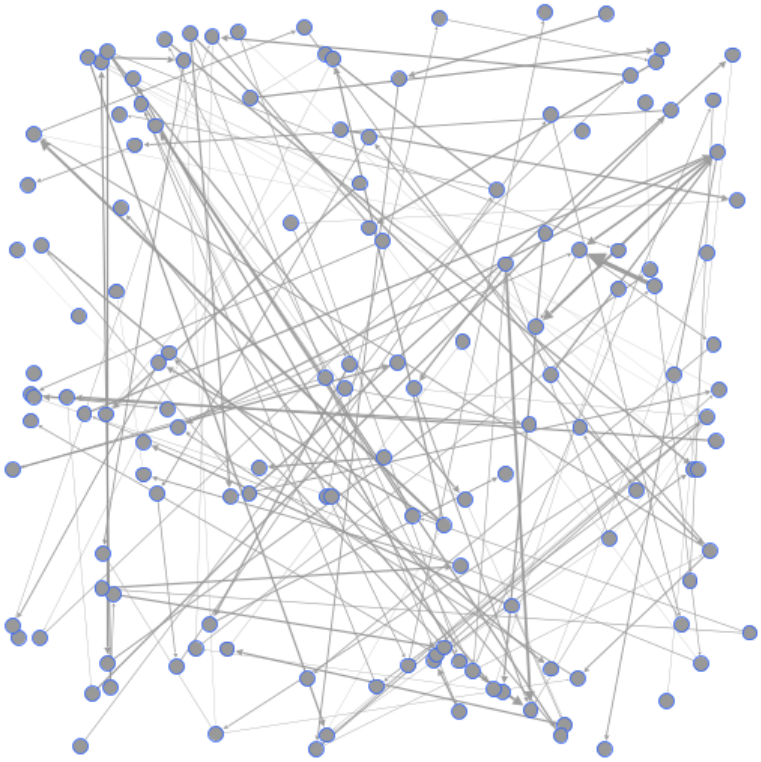


Figura 4.10: Grafo de conceptos asociado al documento 4.2.

en la métrica *betwenness centrality* (4.4.2). También ocultamos las etiquetas y hacemos un poco más gruesas las aristas. Después de ejecutar obtenemos un grafo como el de la figura 4.11 y 4 *clusters*.

4.7.3. Generación de resúmenes

Queda el último de los pasos. Como a la hora de procesar el documento hemos elegido que tendrá una compresión del 30 %, el número de frases que se elegirán serán en torno a 3. Pero antes, se deben asignar las oraciones a los *clusters* (4.6), obteniendo como resultados de similitud (que recordemos son los que permiten hacer el sistema de votación) los mostrados en la figura 4.12.

Para terminar, se seleccionan las oraciones en base a la heurística seleccionada (4.6), en nuestro caso la número 1, que escoge del *cluster* con mayor número de nodos todas las frases. Si miramos la figura 4.12, podemos ver que las oraciones seccionadas pertenecerán al *cluster* 2, y éstas serán la oración 0, la 1 y la 4, por ser las que mayor similitud tienen con dicho *cluster*. El resumen final se puede consultar en el cuadro 4.3.

Comprobaremos también que resumen se hubiese generado si en vez de la heurística 1, hubiésemos escogido la heurística 3. En este caso se debe calcular una puntuación para cada frase, siguiendo la ecuación 4.9, obteniendo la figura 4.13. De ella escogeremos las 3 frases con mayor puntuación, siendo la 0, la 4 y la 1. Obtenemos un resumen idéntico con ambas heurísticas en este caso. Como son documentos pequeños, hay pocas diferencias, a menos que cambiásemos la forma de generar los *clusters* o la métrica usada para valorar

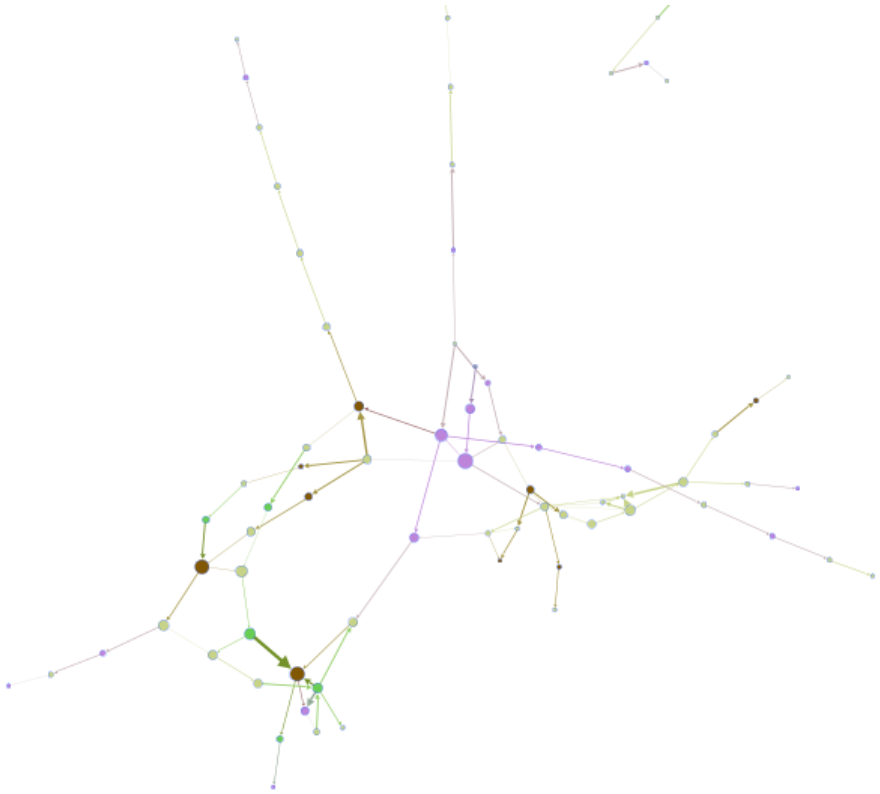


Figura 4.11: Aplicación del *layout* Yifan Hu y la métrica *betwennes centrality* (sólo se muestra una parte del grafo global).

Elem. *Cluster*	Nº *cluster*	or. 0	or. 1	or. 2	or. 3	or. 4	or. 5	or. 6	or. 7	or. 8
32	0	682.5	193.0	54.0	91.0	297.5	40.0	51.0	92.0	0.0
10	1	415.5	170.5	0.0	0.0	328.5	15.0	0.0	0.0	5.0
73	2	1096.46	441.97	93.0	22.0	360.99	188.98	24.0	26.0	30.0
17	3	311.5	136.5	23.0	10.0	115.0	0.0	7.0	7.0	0.0

Figura 4.12: Obtención de la similitud de cada frase con los *clusters* usando la métrica *betwennes centrality*.

```
Resumen Heurística 1:
0.- The goal of the trial was to assess cardiovascular mortality and morbidity
for stroke, coronary heart disease and congestive heart failure, as an
evidence-based guide for clinicians who treat hypertension.
1.- While event rates for fatal cardiovascular disease were similar, there was
a disturbing tendency for stroke and a definite trend for heart failure to occur
more often in the doxazosin group, than in the group taking chlorthalidone.
4.- There was, however, no action taken by either the United States Food and Drug
Administration (FDA) or the Joint National Committee on Prevention, Detection,
Evaluation, and Treatment of High Blood Pressure, which authored the most recent
advisory guideline for the National Heart Lung and Blood Institute [3].
```

Tabla 4.3: Resumen generado del documento 4.2 usando la heurística 1 y la métrica *betwenness centrality*.

Oración	Puntuación
0	96.22
1	37.16
2	4.31
3	3.73
4	53.85
5	5.34
6	2.33
7	3.64
8	0.91

Figura 4.13: Obtención de la puntuación de cada frase.

```
Resumen Heurística 1:
0.- The goal of the trial was to assess cardiovascular mortality and morbidity
for stroke, coronary heart disease and congestive heart failure, as an
evidence-based guide for clinicians who treat hypertension.
1.- While event rates for fatal cardiovascular disease were similar, there was
a disturbing tendency for stroke and a definite trend for heart failure to occur
more often in the doxazosin group, than in the group taking chlorthalidone.
5.- In May 2001, however, after initiation of a class action lawsuit and a
Citizens Petition, the FDA held a hearing on the issue.
```

Tabla 4.4: Resumen generado del documento 4.2 usando la heurística 1 y la métrica *salience*.

los nodos. Esto también repercute a la hora de la selección de oraciones, siendo la más rápida la de la heurística 1 por no tener que hacer cálculos adicionales y agrupar bastante bien el mayor contenido del documento.

Probando a realizar los *clusters* basados en la métrica *salience* (4.4.2.1), obtenemos resultados ligeramente distintos: 6 *clusters*, con lo que la información se distribuye un poco más, con lo que dependiendo de la heurística escogida, puede dar resultados a distintos resúmenes.

Con la heurística 1 seleccionada, el resumen obtenido es el del cuadro 4.4. Se puede analizar cómo éste es un resumen de peor calidad, ya que se ha escogido una frase que tiene que ver con una anterior (la 5 con la 4) que no ha sido incluida. Si por el contrario, usamos la heurística 3, obtenemos el mismo resumen que con la métrica anterior.

Podemos concluir por tanto cómo el usar una métrica u otra afecta directamente a la distinta generación de *clusters* (usando el sistema de generación de *clusters* basados en una métrica), que afectan directamente al resumen final. La heurística 3 es la más completa de las presentadas, siendo una mezcla de la 1 y la 2 y obteniendo mejores resultados en nuestras pruebas. Por contra hay que realizar cálculos adicionales para determinar la importancia de cada frase.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

El proyecto que empezamos a desarrollar a principio del curso ha evolucionado con nosotros. Tuvimos que modificar algunos de los objetivos (2) que nos planteamos al comienzo del curso, pero hemos podido completarlos, implementando un sistema capaz de procesar la información de un informe médico, creando una representación visual de la misma con la que se puede trabajar para determinar que partes son más importantes de una manera rápida e intuitiva a partir de grafos gracias a la herramienta Gephi. También el sistema es capaz de generar un resumen a partir de dicha información. Detallamos objetivos completados a continuación:

- Procesamiento de los grafos conceptuales para la visualización de la información más significativa: Una de las partes más importantes de nuestro proyecto era la de permitir procesar los grafos para poder obtener información sobre los conceptos y detallarlo de manera visual. Lo hemos completado, con resultados bastante vistosos.

- Recuperación de la información procesada en el punto anterior para la generación de resúmenes: Es posible generar resúmenes con varias opciones de configuración, incluyendo distintas formas de generarlo o la longitud máxima que debe tener respecto al tamaño original.

- Mejorar el acceso y la recuperación de la información sobre las relaciones entre conceptos: Hemos mejorado los tiempos de acceso y recuperación de la información integrando estructuras de datos más rápidas y eficientes.

- Proporcionar distintas opciones para modificar la distribución visual de nodos y aristas de un grafo: El sistema es capaz de ofrecer distintas formas de distribuir los nodos y aristas a través de los *layouts*. Además, tiene distintas opciones de configuración, así como opciones predefinidas y múltiples combinaciones posibles.

- Estudio de distintas métricas para hacer agrupaciones de nodos significativos: Se ha conseguido integrar tanto una de las métricas usadas en la tesis de referencia [16], como nuevas ofrecidas por *Gephi*.

- Permitir distintas configuraciones para modificar la apariencia visual de un grafo: El sistema ofrece gran cantidad de opciones para configurar la apariencia de un grafo. Se pueden mostrar u ocultar etiquetas, cambiar los bordes de etiquetas y nodos, tamaños de fuente, mostrar u ocultar aristas y cambiar su tamaño. También se puede modificar la opacidad de nodos y aristas.
- Aumentar la interacción, resaltando la información de los nodos que se seleccionen, así como la recolocación manual de los mismos: Aunque sigue siendo de forma limitada, hemos conseguido aumentar la interacción, permitiendo la selección, desplazamiento y borrado de los nodos.
- Permitir el filtrado por importancia de nodos o grupos significativos de nodos: El sistema es capaz de filtrar la información visualmente de forma que se pueden mostrar sólo los nodos contenidos en un rango delimitado por una escala del 1 al 5 en importancia, mostrando además a qué *clusters* pertenecen. También puede hacer que sólo se muestre uno de los *clusters* que seleccione el usuario.

5.2. Ampliaciones y Trabajo futuro

El sistema desarrollado puede ser usado como base para otro tipo de aplicaciones que necesiten trabajar con información en forma de grafos, aunque también se puede ampliar con mejoras que aumentarían la funcionalidad y la capacidad del sistema que detallaremos en los siguientes puntos.

5.2.1. Resúmenes Multi-documento

Uno de los objetivos que tuvimos que modificar es el de generar los resúmenes multi-documento. Sin embargo, siguiendo la información de Plaza [16], es perfectamente factible ampliar esta funcionalidad, consiguiendo que el proyecto aumente mucho en interés por la comunidad médica debido a la gran síntesis que se haría de toda la información que tienen que manejar. Para ello habría que modificar la etapa del preprocesamiento (para poder procesar varios informes médicos y generar varios grafos) y adaptar las etapas de visualizado de grafos (nuevas opciones de filtrado por documentos) y de generación de resúmenes (evitar ciertos problemas asociados a la generación de un resumen con varios documentos como es el de la redundancia).

5.2.2. Resúmenes de calidad

El actual sistema desarrollado resume el contenido de un informe seleccionando un número de oraciones que considera las más importantes, que son las que se seleccionan y aparecen en el resumen final. Los resúmenes generados por tanto son de una calidad bastante pobre, haciendo que en ocasiones no sigan una coherencia o se refieran a algo que no se ha nombrado. Este problema sería aún mayor si se hiciesen resúmenes multi-documento. Para solucionar este problema, encontramos ideas en la tesis de Plaza [16], en dónde se explica la teoría para implementar esta funcionalidad. Aplicando controles de redundancia, haciendo un buen uso de la elipsis y reescribiendo oraciones para sintetizar varias en una sola, se podría obtener un resumen de muy alta calidad.

5.2.3. Mejorar las consultas a la base de datos

Investigar más en profundidad `UMLS` para elegir aquellas fuentes y acotar el ámbito de búsqueda en la base de datos. `SNOMEDCT` es uno de los principales que hemos utilizado, pero hay otros campos llamados `SAB`, en `UMLS`, que pueden ser interesantes y otros que no se deberían incluir en la base de datos de cara a mejorar las consultas, lo que mejoraría el rendimiento y la calidad de los resúmenes generados.

5.2.4. Otros idiomas

Ampliar el proyecto para poder trabajar con más idiomas. Actualmente puede procesar únicamente informes médicos en inglés y la interfaz está en el mismo idioma. Podrían añadirse opciones de configuración para que la interfaz estuviese en más idiomas y que pudiese trabajar con informes en otros idiomas. Sin embargo, esto último deberá ser un objeto de estudio en sí mismo, debido a que el procesamiento de la información en otros idiomas está menos avanzado que el inglés, al igual que las bases de conocimiento de las que se obtiene la información adicional.

5.2.5. Mejorar la precisión de la información

Actualmente se disponen de un número de métricas para trabajar con nodos y dos formas de generar los *clusters*. Puede ser interesante aumentar estos números para realizar más pruebas con el objeto de identificar con más acierto los conceptos más significativos del documento así como las frases más importantes. Por otro lado, mejorar el procesamiento de los grafos añadiendo o incluso creando *layouts* personalizados podrían mejorar las distribuciones de los nodos de una manera mucho más interesante.

5.2.6. Aumentar las posibilidades de interacción con el usuario

Algunas ideas de interacción entre los grafos y el resumen se quedaron en el camino debido a la falta de tiempo. Nos gustaría sugerirlas para su desarrollo en un futuro porque creemos que añadiría más profundidad al sistema:

- Al seleccionar los nodos del grafo que también se indique la frase en el documento origen.
- Colorear las frases del resumen generado como en el grafo para que sea más visual ver la relación en ellos.
- Modificar el grafo resumen y que se modifique al mismo tiempo el resumen texto (actualmente se permite modificar ligeramente los grafos pudiendo eliminar algunos nodos, pero es algo bastante básico respecto a la idea inicial).

Apéndice A

Manual de Usuario

A.1. Requisitos previos

Para poder ejecutar la aplicación que hemos desarrollado es necesario cumplir una serie de requisitos software previos. Necesitaremos tener instalados los siguientes componentes:

- *Java Runtime Environment* o *JRE* de *64 bits*, versión 1.7 o superior.
  Para ello desde la página de Oracle recomiendan, para plataformas de 64 bits un procesador mínimo Pentium 2 a 266 MHz. Se necesitará un espacio mínimo de 181MB. En lo referente a la memoria, para plataformas de 64 bits se necesitará una RAM mínima de 128MB para cualquier sistema operativo, se recomienda usar más RAM para aplicaciones que se ejecuten sin un explorador que use el plugin de Java. Ejecutarlo con menos memoria de la recomendada puede causar un *swap* en el disco, lo cuál tiene efectos severos en el rendimiento.
  Enlace de descarga: `http://www.java.com/es/download/`.

- *MetaMap*.
  Es un programa desarrollado por la *National Library of Medicine* de los Estados Unidos para la traducción de textos biomédicos a conceptos del Metatesauro de `UMLS`. Hemos usado la versión 2012 para el desarrollo de la aplicación.
  Enlace de descarga: `http://metamap.nlm.nih.gov/#Downloads`
  Explicamos cómo se lleva a cabo la correcta instalación de *MetaMap* en el apéndice de Instalación y ejecución de MetaMap y MetaMap Java API, para ello habrá que tener una cuenta de `UMLS`, se puede solicitar en el siguiente enlace: `https://uts.nlm.nih.gov//license.html`.

- *Base de datos*.
  Se necesitará tener descargada la base de datos de `UMLS` para poder realizar las consultas necesarias para la extracción de la información. Como la base de datos de `UMLS` es muy grande y pesada hemos trabajado con una parte de la base de datos solamente sacada con *MetamorphoSys*. Explicaremos en el siguiente apéndice cómo crear la base de datos.

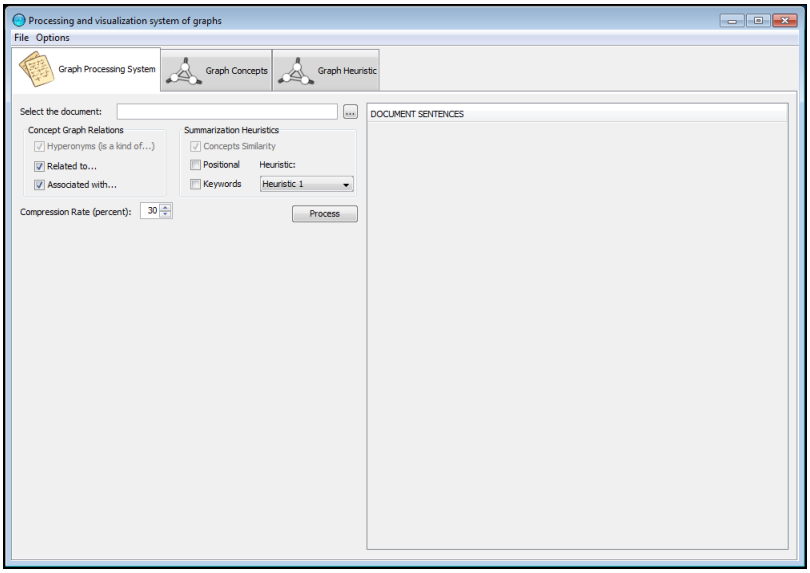


Figura A.1: Apariencia de la aplicación al iniciar posicionada en la primera pestaña.

A.2. Ejecución y uso del sistema

En esta sección del capítulo explicaremos el funcionamiento de nuestra aplicación, el flujo de trabajo y las distintas opciones que la componen.

La interfaz de nuestra aplicación consta de un panel principal con *JTabbedPane* con tres pestañas, una por cada uno de los paneles que requiere nuestra aplicación. A continuación explicaremos la utilidad y funcionamiento de cada uno de los paneles en el orden en que deben de ser usados en nuestra aplicación. La figura A.1 muestra la aplicación al arrancar.

A.2.1. Barra de menús

En nuestra aplicación tenemos un *JMenuBar* con los menús *File* y *Options*, en el menú *File* podemos cargar un grafo con extensión `.gexf` ya generado para visualizarlo y poder interactuar con él desde la pestaña *Graph Concepts* (véase A.2.3), el menú *Options* se puede abrir un JDialog para configurar las opciones avanzadas para el procesado de los grafos. Estas opciones pueden verse en la subsección Configuración avanzada.

A.2.2. Pestaña `GPS`

La pestaña `GPS`[1], la primera que aparece al iniciar la aplicación sirve para procesar los informes médicos con los que trabajaremos. Tiene asignada *ALT+1* como atajo de teclado para moverse a esta pestaña. Se ve como en la figura A.2.

- El botón "..." al lado del campo de texto sirve para cargar el informe con extensión `.xml`, el campo de texto no se puede modificar manualmente, hay que seleccionar el

[1] `GPS`: *Graph Processing System* (Sistema de Procesamiento de Grafos).

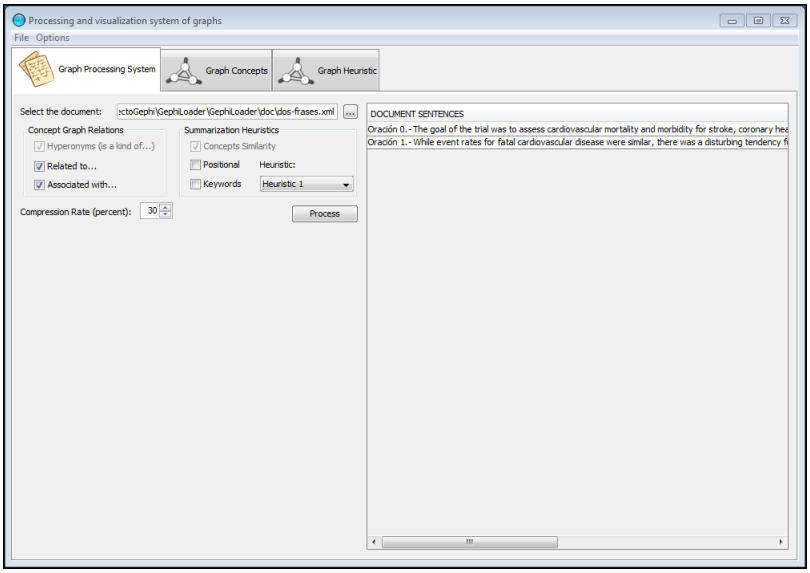


Figura A.2: Apariencia de la pestaña GPS al procesar el archivo de ejemplo *dos-frases.xml*.

archivo mediante el botón, esto está así para prevenir que se introduzca mal la ruta del archivo a procesar.

- El panel *Concept Graph Relations* hay tres CheckBox que indican las relaciones que queremos coger entre los conceptos del documento, la relación de hiperonimia está marcada por defecto y no se puede desactivar, esto solo se muestra para que se vea que aparecerá la relación. Las opciones *Related to...* y *Associated with...* darán diferentes relaciones y podrán mostrar más conceptos del informe a procesar.
- El panel *Summarization Heuristics* tienen las opciones que usarán las heurísticas al generar el resumen del informe mediante el grafo de conceptos.
- *Compression Rate* indica el porcentaje de compresión del informe a la hora de generar el resumen, por defecto está determinado que el ratio de compresión del informe sea del 30 %.
- El botón *Process* procesa el archivo indicado en el campo de texto.
- La tabla con cabecera *Document Sentences* mostrará todas las frases procesadas del informe una vez haya sido procesado.

A.2.3. Pestaña Graph Concepts

La pestaña *Graph Concepts*, será donde se muestre el grafo principal de conceptos extraídos del informe y con el que se pueda interactuar. Tiene asignada *ALT+2* como atajo de teclado para moverse a esta pestaña. Se ve como la figura A.3 muestra los diversos componentes del panel del grafo de conceptos.

- El área de texto *Information* muestra los conceptos extraídos del informe mediante *MetaMap* que van a ser usados en el grafo de conceptos. Si al grafo se le ha aplicado

un cálculo de *clustering*, a cada nodo le será asignado un color correspondiente al *cluster* que le corresponda. El panel principal de esta pestaña contiene un *JSplitPane* por lo que puede ampliarse arrastrando el separador a la derecha, minimizarse hacia la izquierda y con las dos flechitas arriba del separador ocultar o maximizar el área de información.

- En la ventana del grafo se representará el grafo de conceptos del informe con el cuál podremos interactuar. Los nodos del grafo representan los conceptos y están etiquetados con los nombres de estos. Mediante los botones de la barra de herramientas podremos interactuar con el grafo.
- Barra de herramientas
  - En el primer separador hay dos iconos: el primero  sirve para ejecutar la configuración seleccionada para el grafo (forma, puntuación, si se aplican *clusters*, etc.) y el segundo  para una vez ejecutada la configuración seleccionada, generar un resumen y el grafo del resumen que se podrán ver en la pestaña *Graph Heuristic*.
  - Entre el primer y segundo separador hay dos iconos de opciones: el primero  sirve para seleccionar opciones de filtrado en caso de que queramos filtrar los nodos del grafo mediante la puntuación dada al ejecutar la configuración o mediante el *cluster* deseado eligiéndolo por color; el segundo icono  sirve para mostrar el panel de opciones de la figura A.3, seleccionando uno u otro *layout*, *ranking* u opciones de *clustering* podemos configurar de una manera rápida y predeterminada la configuración para el grafo. Para más opciones mirar Configuración avanzada.
  - Entre el segundo y tercer separadores hay tres iconos para interactuar con los nodos del grafo, el primero  permite seleccionar un nodo del grafo y ver su etiqueta en la propia barra de tareas, esto es útil sobre todo cuando hay muchos nodos y las etiquetas no se pueden ver todas bien en el grafo; el segundo icono  , la cruz azul sirve para mover los nodos del grafo, seleccionar un nodo con la cruz te permitirá hacer *click* en otro lugar de la ventana del grafo para mover ahí el nodo seleccionado; por último la equis roja  que sirve para eliminar nodos del grafo.

A.2.4. Pestaña Graph Heuristic

La pestaña *Graph Heuristic*, será donde se muestre el grafo del resumen y el resumen correspondiente. Con este grafo no se puede interactuar, sirve para comparar con el re-

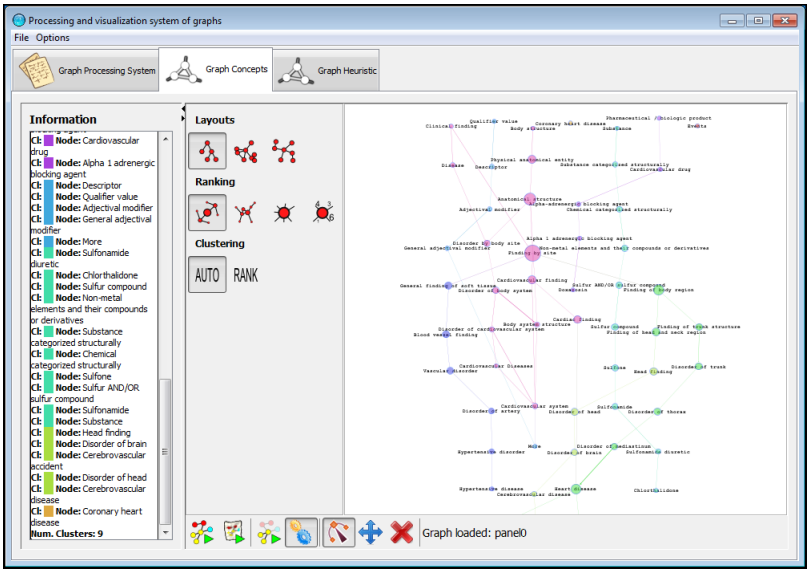


Figura A.3: Apariencia del Panel Graph Concepts al haber procesado un informe y haber ejecutado los clústers con una organización del grafo en árbol.

sumen generado por el grafo de conceptos de la pestaña *Graph Panel*. Tiene asignada *ALT+3* como atajo de teclado. Se ve cómo la figura A.4 muestra el resumen y el grafo correspondientes.

A.2.5. Configuración avanzada

En el menú de opciones podemos abrir el panel de opciones avanzadas para el grafo de conceptos. Este panel consta de tres partes:

- Pestaña *Ranking*: como la figura A.5, aquí podremos elegir el tipo de métrica usado a la hora del cálculo de puntuaciones para los conceptos. El tipo o los tipos de layouts (disposición visual de los nodos en el grafo) que queramos usar, si queremos forzar a que las etiquetas de los nodos no se superpongan unas con otras y el tipo de ejecución para el cálculo de las puntuaciones del grafo y la correcta colocación del layout. A más tiempo se obtendrán mejores resultados.

- Pestaña *Configuración*: como la figura A.6, aquí podremos elegir opciones para modificar los distintos elementos que componen un grafo: nodos, etiquetas y aristas. Para los nodos podremos modificar su tamaño, el ancho del borde, la opacidad del relleno y el color del borde; en las etiquetas podremos decidir si son visibles o no, si se pueden redimensionar, meter en cajas, cortar las etiquetas muy largas, el tamaño de la fuente, el color, el número de caracteres máximo permitido y el grosor de las letras; para las aristas podemos mostrarlas u ocultarlas, ajustar los pesos, hacer aristas curvas o rectas, la opacidad de éstas y el grosor de la arista así como de la punta de flecha.

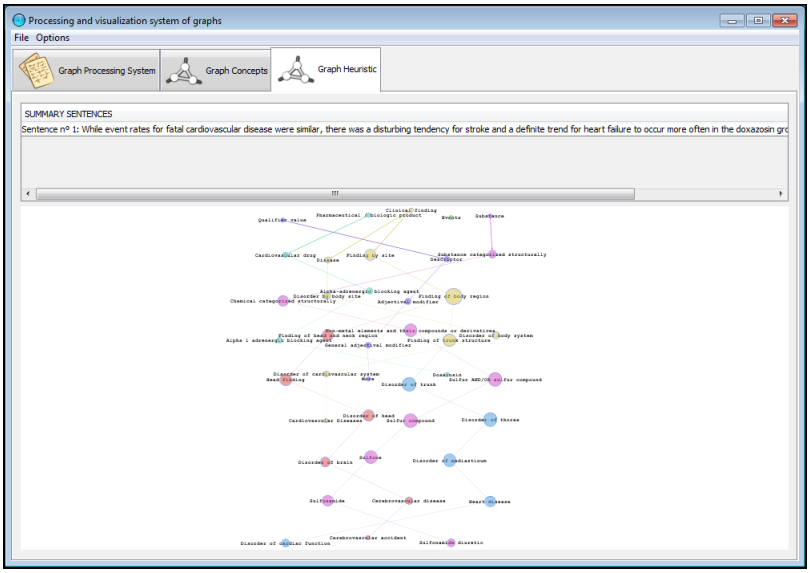


Figura A.4: Apariencia del Panel Graph Heuristic al haber generado el resumen desde el Panel Graph Concepts.

- Pestaña *Graph Concepts: Clustering*: como la figura A.7, aquí simplemente podemos decidir si queremos que se calculen o no los clústers al ejecutar la cofiguración del grafo y que se cojan solo los que devuelvan determinados valores.

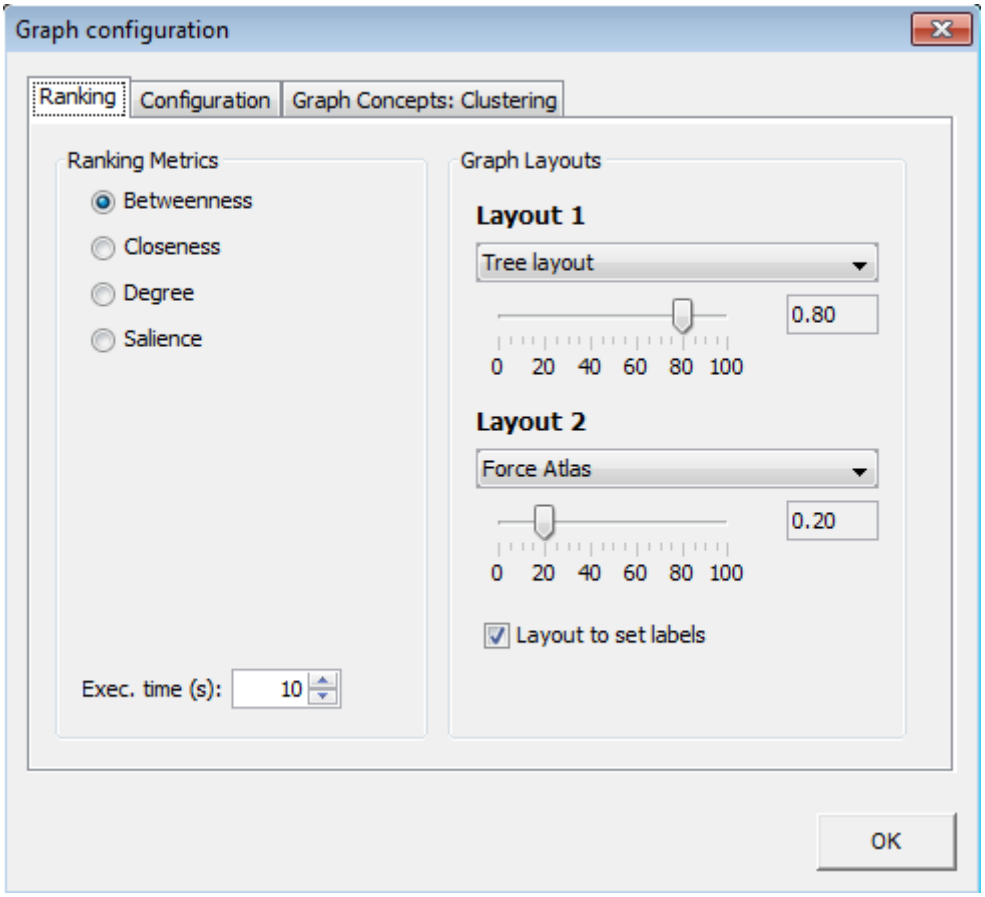


Figura A.5: Pestaña para elegir la configuración de las métricas y los layouts.

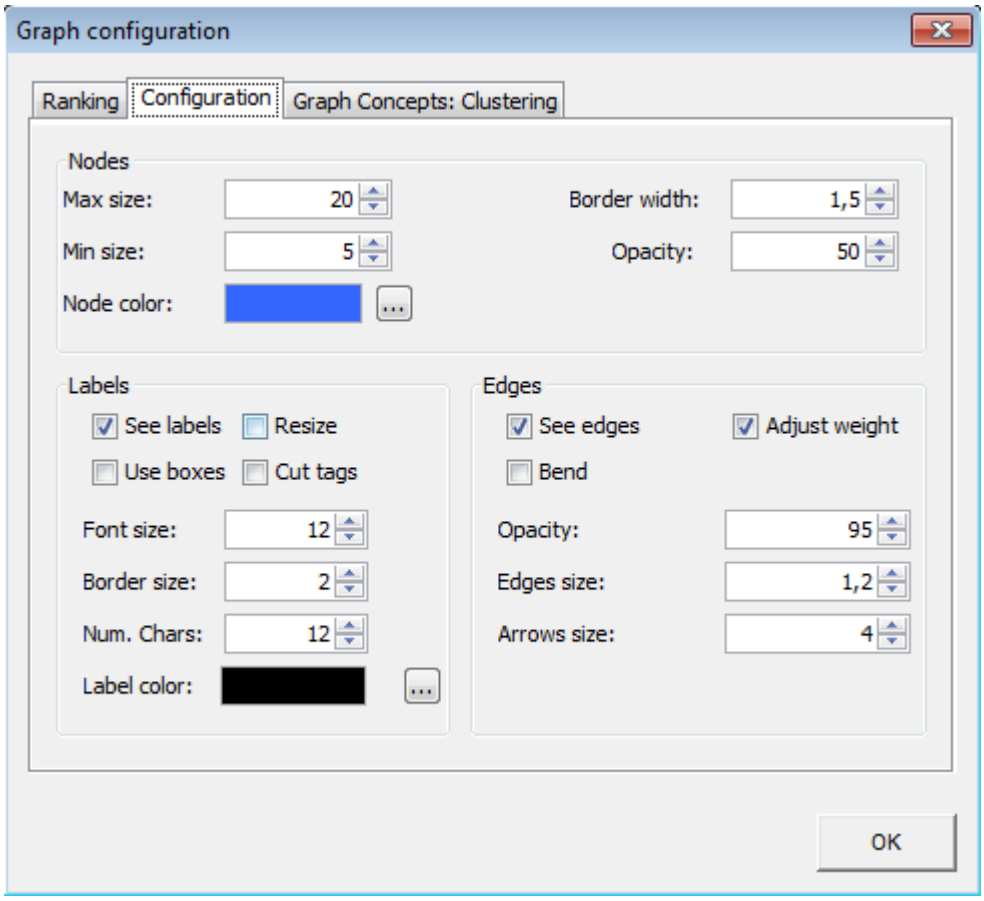


Figura A.6: Pestaña de varias opciones para modificar los elementos del grafo.

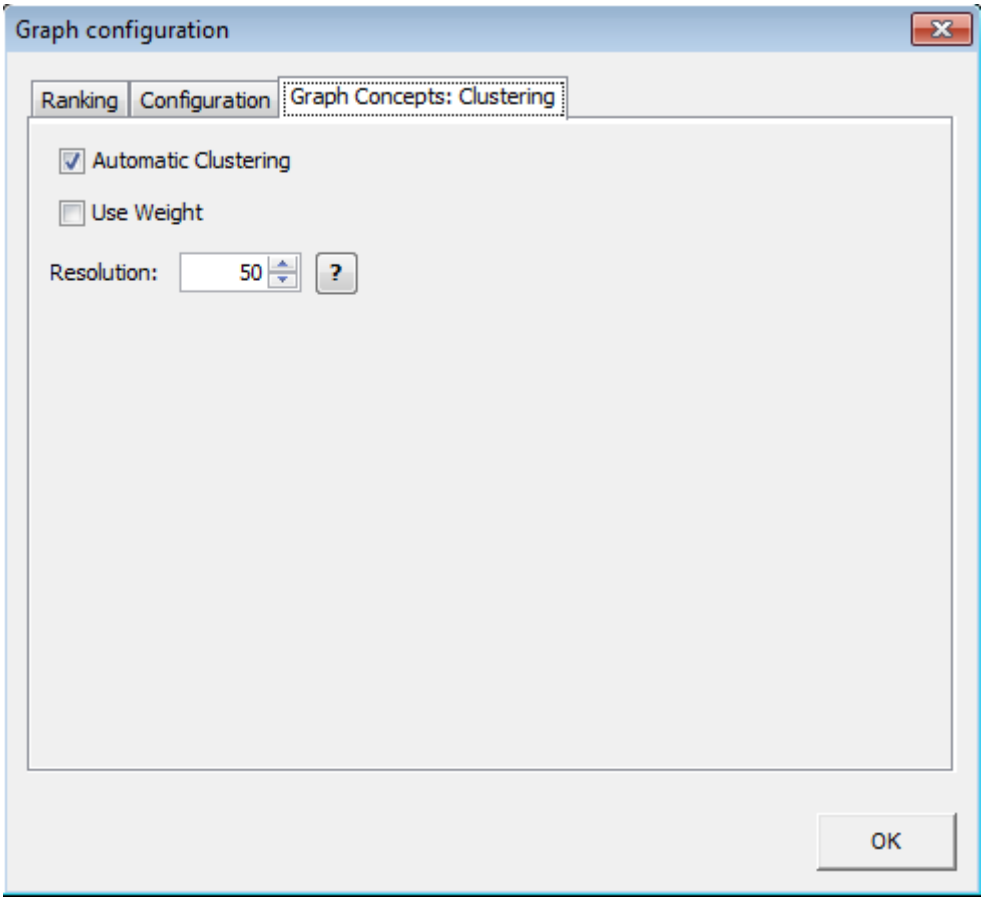


Figura A.7: Pestaña para la configuración de los clústers.

Apéndice B

Instalación y ejecución de MetaMap y MetaMap Java API

En este apéndice explicaremos cómo realizar la instalación de *MetaMap* y de *MetaMap Java API*.

B.1. Requisitos previos

Antes de poder instalar *MetaMap*, hay que descargarlo de la web. Nosotros hemos usado la versión 2012.

1. El primer paso que hay que hacer es registrarse en `UMLS` para poder descargarlo, el registro se realiza desde: `https://uts.nlm.nih.gov//license.html`, el registro en `UMLS` no es inmediato, el registro debe de ser aprobado y aceptado mediante un correo notificándolo, esto puede tardar algunos días.
2. Una vez estemos ya registrados, tendremos que descargarnos la versión que queramos descargar según nuestro sistema operativo, lo podemos descargar en el siguiente enlace: `http://metamap.nlm.nih.gov/#Downloads`.
3. También hay que descargar *MetaMap Java API* desde: `http://metamap.nlm.nih.gov/#MetaMapJavaApi`.
4. Es conveniente guardar ambos archivos comprimidos en una misma carpeta `MetaMap`. Para evitar problemas es mejor que en la ruta de la carpeta no haya espacios ni símbolos de puntuación o acentuación.
5. Para poder realizar la instalación hay que tener una versión de *Java* instalada, si está en la ruta por defecto no habrá que indicarla después durante la instalación.

B.2. Instalación de MetaMap

Hay que descomprimir los dos archivos comprimidos en la carpeta `MetaMap` que hemos creado donde se extraerá una carpeta llamada `public_mm`, primero *MetaMap* y luego *MetaMap Java API* sobrescribiendo los archivos con nombres iguales. Ahora habrá que lanzar

unos *scripts* según estemos en *Windows* o *Linux*. En algunas versiones dentro de los *scripts* se indica que se carga el directorio `bin` y luego el *script*, si ya se está en `bin` fallará por lo que habría que ejecutarlo desde el nivel superior del directorio.

B.2.1. Instalación en *Windows*

Hay que abrir la carpeta `public_mm` que se ha creado y ejecutar el `.bat`: *Install MetaMap.bat*. El script según la versión de *MetaMap* puede haber un problema de rutas, el contenido del script sería el siguiente:

```
set PATH=%PATH%:%CD%\bin
set PWD=%CD%
bin\wish bin/ginstall.tcl
```

Ese *script* lanza una interfaz gráfica para guiar en la instalación de *MetaMap*. Solicita la ruta en la que se desea instalar *MetaMap* y donde se encuentra la versión de *Java* (si se instaló en la ubicación por defecto la encontrara automáticamente).

B.2.2. Instalación en *Linux*

Nos ponemos en *MetaMap/public_mm/bin* y ejecutar el *script install.sh* desde el terminal o haciendo doble click en el archivo.

Apéndice C

Crear una ontología personalizada

UMLS contiene gran cantidad de información que puede interesarnos una parte o algunas fuentes en particular. Debido al gran tamaño también puede ralentizar el proceso de obtener los conceptos y las relaciones ya que debe acceder a mucha más información. Por todo esto, vamos a explicar cómo crear una base de datos con las fuentes que nosotros elijamos utilizando la herramienta *MetamorphoSys*.

C.1. Creación de una cuenta UMLS

Para poder utilizar y realizar consultas a UMLS hay dos formas de hacerlo. Una es descargarnos UMLS y crearnos la base de datos para que nuestros proyectos puedan acceder a la información por medio de consultas sql utilizando la herramienta *MetamorphoSys*. Otra que también es muy útil es la de UMLS interactivo, que nos permite hacer consultas directamente a UMLS a través del navegador consiguiendo así la información que necesitamos. Esta forma de consulta también nos servirá para comparar los resultados y nos permite elegir cualquier versión de UMLS, que por otro lado, con la opción de *MetamorphoSys* debemos descargarnos dicha versión. Elegiremos la opción que más nos convenga, en nuestro caso elegimos instalar la base de datos ya que haremos un uso intensivo de ella para generar nuestros resúmenes.

Tanto para usar el UMLS interactivo como para descargarnos UMLS y *MetamorphoSys* debemos registrarnos en la página de UMLS.https://utslogin.nlm.nih.gov/cas/login

Nos creamos una cuenta con unos sencillos pasos:

- Accedemos a la página de UMLS y hacemos click en *Request one now* como se ve en la Figura C.1.

- Aceptamos las licencias y condiciones del *Metathesaurus* UMLS .

- Rellenamos un pequeño cuestionario con los datos personales y el propósito de la cuenta. En nuestro caso nos registramos como alumnos de la Universidad Complutense.

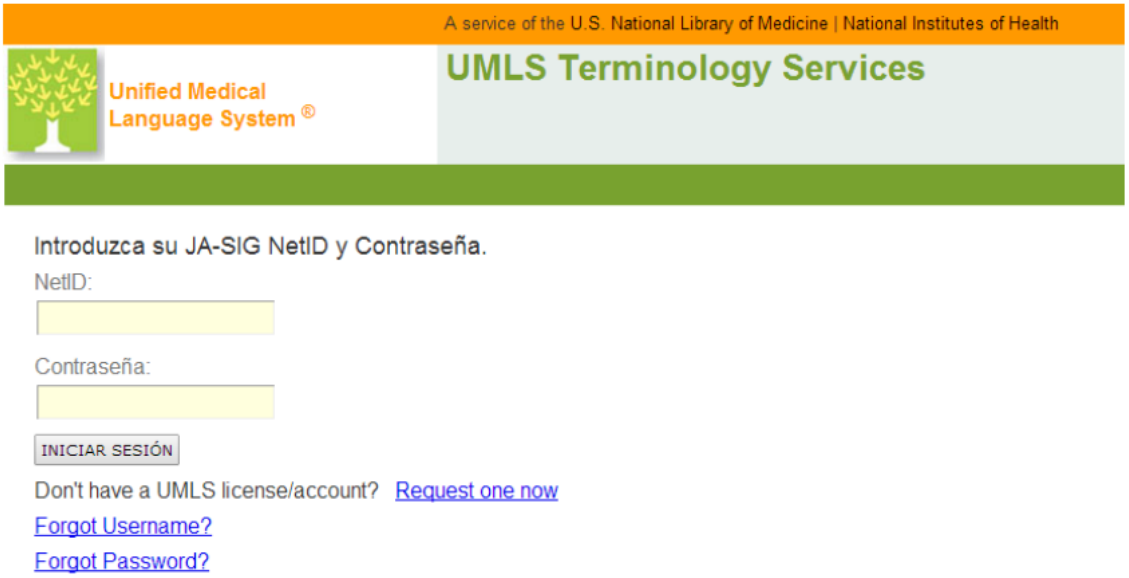


Figura C.1: Pantalla de *Login*

- Después de haber completado el formulario se enviará la petición y tendremos que esperar unos días para que la petición sea aceptada.

- Recibiremos en nuestro correo la confirmación de la solicitud y nos permitirá iniciar sesión en `UMLS` para poder utilizar `UMLS` interactivo o descargarnos la base de datos.

C.2. `UMLS` *Metathesaurus* interactivo

Una vez iniciado sesión podemos utilizar el `UMLS` interactivo que nos permitirá hacer pequeñas consultas de conceptos y nos mostrará la información completa de dicho término, tales como las relaciones, jerarquía, sinónimos, etc. Se encuentra en `https://uts.nlm.nih.gov/metathesaurus.html`.

Nos permite configurarlo a nuestras necesidades y nos da la opción de elegir la versión y si queremos restringir la búsqueda a un subconjunto del Metatesauro. Podemos realizar la búsqueda de los conceptos por su código, `CUI` o término.

Ejemplo de *hypertensive disease* en la Figura C.3:

Esta forma es muy útil para la búsqueda de conceptos concretos o cuando se necesita saber algo en particular pero recordamos que si se precisa un uso intensivo de la base de datos, la mejor opción es crearla con *MetamorphoSys* y configurarlo según necesitemos, ya que llamadas a nivel local son mucho más rápidas que hacerlas al servidor.

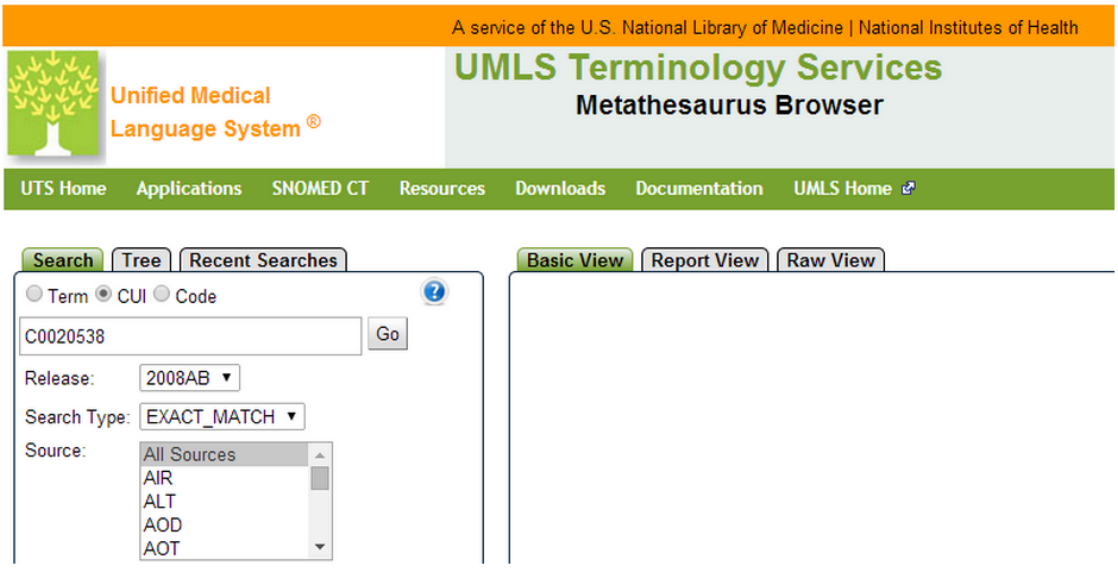


Figura C.2: UMLS interactivo

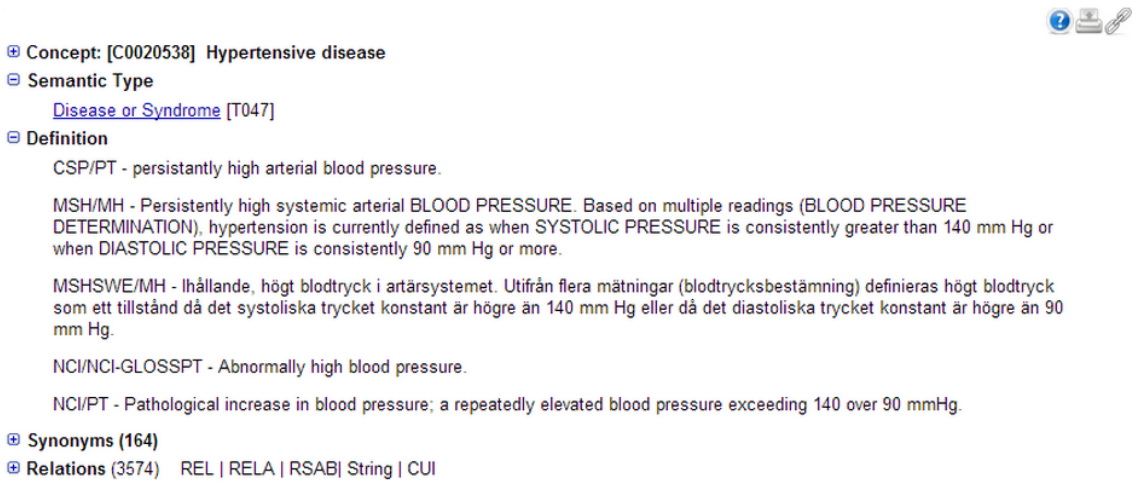
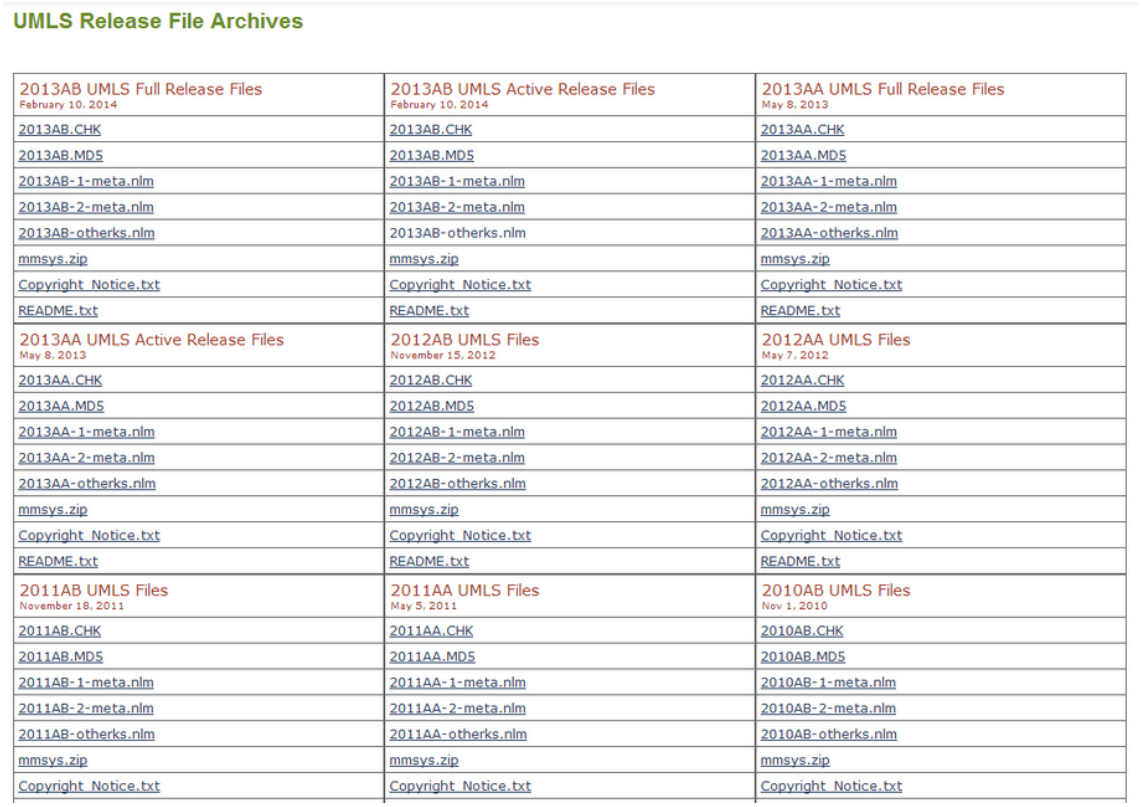


Figura C.3: Ejemplo Hypertensive Disease



UMLS Release File Archives

2013AB UMLS Full Release Files February 10, 2014	2013AB UMLS Active Release Files February 10, 2014	2013AA UMLS Full Release Files May 8, 2013
2013AB.CHK	2013AB.CHK	2013AA.CHK
2013AB.MD5	2013AB.MD5	2013AA.MD5
2013AB-1-meta.nlm	2013AB-1-meta.nlm	2013AA-1-meta.nlm
2013AB-2-meta.nlm	2013AB-2-meta.nlm	2013AA-2-meta.nlm
2013AB-otherks.nlm	2013AB-otherks.nlm	2013AA-otherks.nlm
mmsys.zip	mmsys.zip	mmsys.zip
Copyright_Notice.txt	Copyright_Notice.txt	Copyright_Notice.txt
README.txt	README.txt	README.txt
2013AA UMLS Active Release Files May 8, 2013	2012AB UMLS Files November 15, 2012	2012AA UMLS Files May 7, 2012
2013AA.CHK	2012AB.CHK	2012AA.CHK
2013AA.MD5	2012AB.MD5	2012AA.MD5
2013AA-1-meta.nlm	2012AB-1-meta.nlm	2012AA-1-meta.nlm
2013AA-2-meta.nlm	2012AB-2-meta.nlm	2012AA-2-meta.nlm
2013AA-otherks.nlm	2012AB-otherks.nlm	2012AA-otherks.nlm
mmsys.zip	mmsys.zip	mmsys.zip
Copyright_Notice.txt	Copyright_Notice.txt	Copyright_Notice.txt
README.txt	README.txt	README.txt
2011AB UMLS Files November 18, 2011	2011AA UMLS Files May 5, 2011	2010AB UMLS Files Nov 1, 2010
2011AB.CHK	2011AA.CHK	2010AB.CHK
2011AB.MD5	2011AA.MD5	2010AB.MD5
2011AB-1-meta.nlm	2011AA-1-meta.nlm	2010AB-1-meta.nlm
2011AB-2-meta.nlm	2011AA-2-meta.nlm	2010AB-2-meta.nlm
2011AB-otherks.nlm	2011AA-otherks.nlm	2010AB-otherks.nlm
mmsys.zip	mmsys.zip	mmsys.zip
Copyright_Notice.txt	Copyright_Notice.txt	Copyright_Notice.txt

Figura C.4: Descarga UMLS.

C.3. Descargando e instalando *MetamorphoSys*

Una vez iniciada sesión con nuestra cuenta `UMLS` podemos pasar a descargarnos todo lo necesario para crear nuestra base de datos personalizada. Accedemos a la zona de descargas y elegimos descargar `UMLS` o en el siguiente enlace `http://www.nlm.nih.gov/research/umls/licensedcontent/umlsarchives04.html`.

Se nos abrirá en el navegador una ventana como la Figura C.4.

Como se ve en la imagen nos dan la opción de elegir la versión que nos interesa. En nuestro caso elegimos la `2013AB` y descargamos todos los ficheros correspondientes incluido *mmsys.zip* (*MetamorphoSys*). Unas notas importantes sobre la descarga:

- Descargar todos los datos `UMLS` y extraerlos en el mismo directorio.
- Información adicional en el enlace `http://www.nlm.nih.gov/research/umls`.
- A una velocidad de 1MB/s tardará aproximadamente cuatro horas en descargar.

Algunos requisitos para el correcto funcionamiento son :

- *MetamorphoSys* requiere un mínimo de 30GB libres en el disco duro y entre dos y diez horas ejecutarlo. Dependerá de la máquina, configuración y sistema operativo.

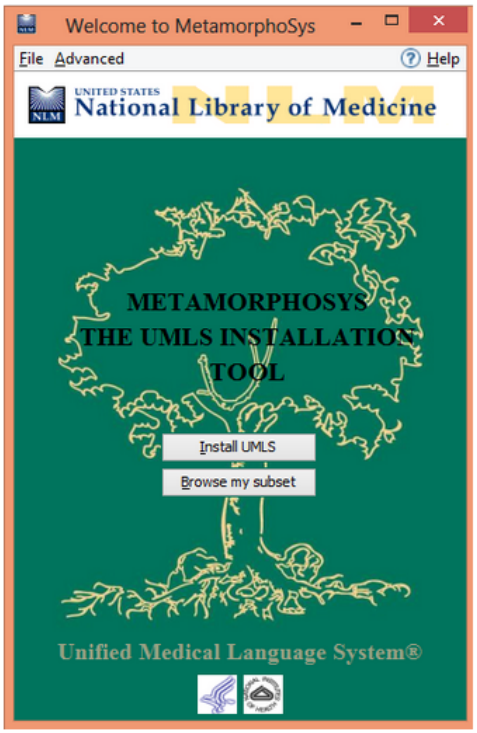


Figura C.5: Metamorphosys incio.

- Mínimo 2GB de RAM. Una memoria más pequeña producirá problemas de paginación, incrementando exponencialmente el tiempo.
- CPU de al menos 2Ghz.
- Asegurarse que todos los ficheros están en el mismo directorio. *-mmsys.zip -2013ab-1-meta.nlm -2013ab-2-meta.nlm -2013ab-otherks.nlm -2013AB.CHK -2013AB.MD5 -Copyright_Notice.txt -Readme.txt*.

Una vez descargados y descomprimidos los ficheros pasamos a la ejecución de *MetamorphoSys*.

- Ejecutamos el *script* correspondiente según estemos en *Windows* o *Linux*. En nuestro caso es *Windows* y ejecutamos `run.bat` que se verá como en la Figura C.5.
- Elegimos el directorio donde tenemos los ficheros descargados y la ruta de destino. Debemos elegir la opción `mysql5.5` para que genere los scripts para *mysql* y con todas las opciones marcadas (*Metathesaurus*, red semántica, lexicón especializado). Se verá como en la Figura C.6.
- Configuramos y elegimos los subconjuntos que deseamos de `UMLS`. Lo veremos como en la Figura C.7.

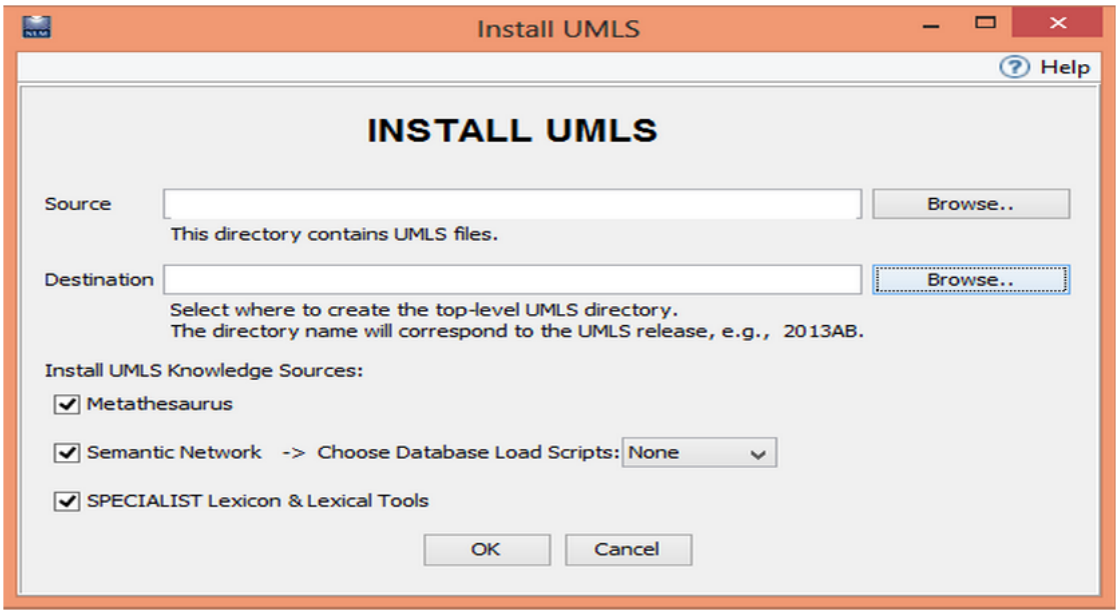


Figura C.6: Instalación Metamorphosys.

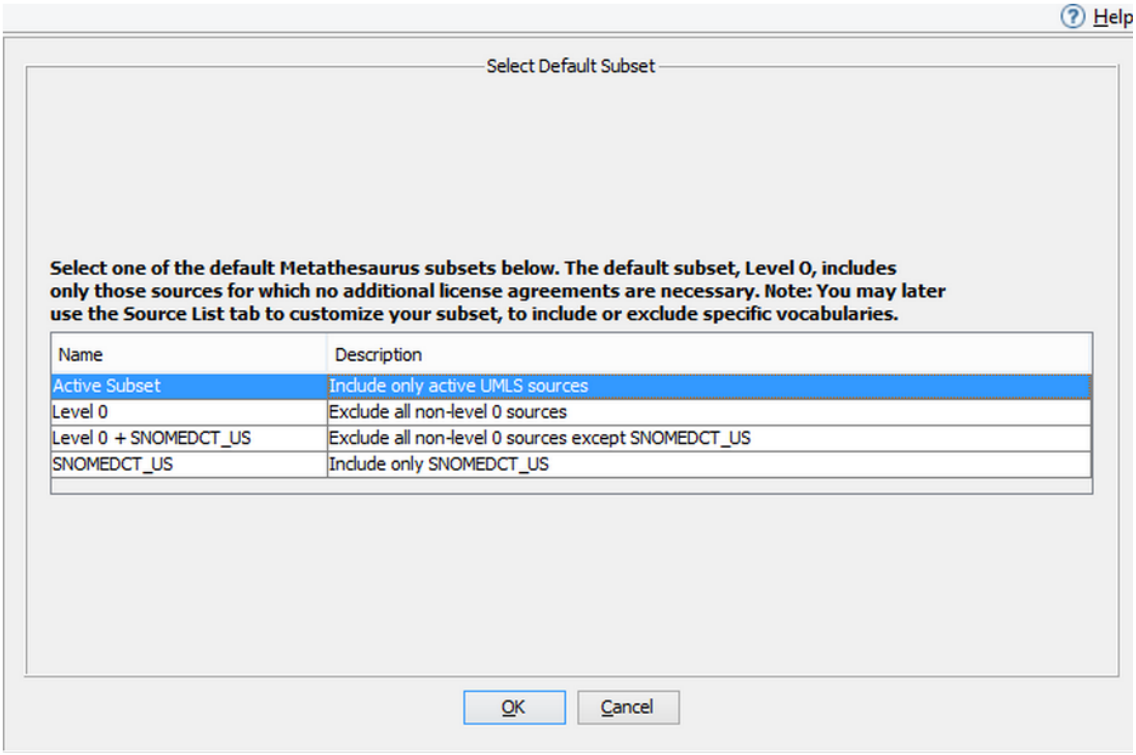


Figura C.7: Selección de los subconjuntos UMLS.

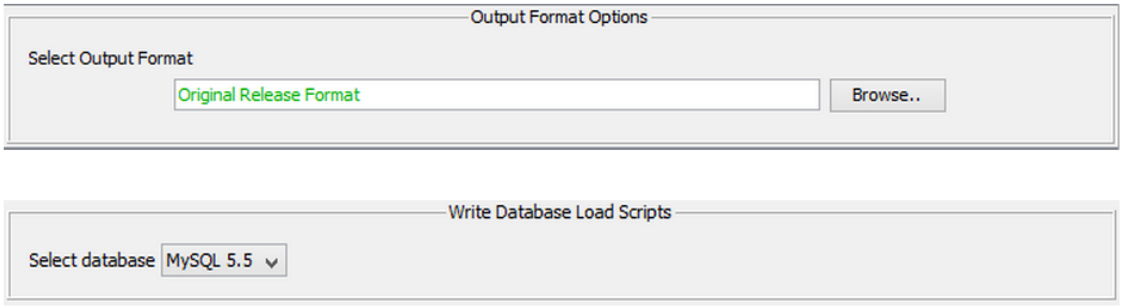


Figura C.8: Configuración de los ficheros de salida.

- Importante elegir en la pestaña *output* salida tipo `ORF` o `RRF` y que genere los *scripts mysql5.5*. `RRF` es la que hemos usado en nuestro proyecto. Véase la Figura C.8.

- Ejecutamos y esperamos a que se complete el proceso. Puede tardar desde varios minutos a algunas horas dependiendo del ordenador. La Figura C.9 muestra la ventana de instalación.

- Una vez terminado el proceso tendremos la carpeta `2013AB` con los *scripts* generados (25 GB aproximadamente con todos los subconjuntos elegidos). Los ejecutamos para instalar la base de datos según indicamos en el Manual de Instalación de la Base de Datos.

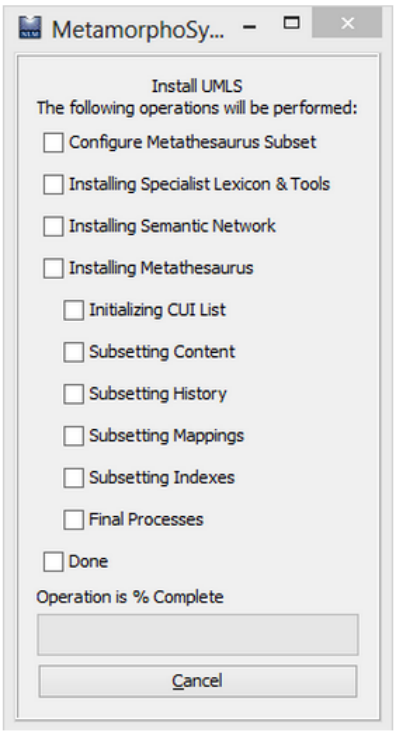


Figura C.9: Creación de la base de datos e instalación de UMLS.

Apéndice D

Manual de Instalación de la Base de Datos

D.1. Requisitos previos

En primer lugar necesitamos tener instalado una versión de *Oracle* o *MySql* en nuestro ordenador. Siguiendo el ejemplo, nosotros hemos generado los *scripts* de la base datos para la versión *mysql5.5* por lo que este capítulo se centrará en ese supuesto. Si aún no disponemos de dicha versión, nos lo podemos descargar en el siguiente enlace http://dev.mysql.com/downloads/mysql/5.5.html#downloads, elegimos la versión de nuestro sistema operativo y descargamos el fichero. Nosotros hemos elegido la versión *Mysql Community Server 5.5.37*. Recomendamos también bajarnos el *Mysql Workbench* http://dev.mysql.com/downloads/tools/workbench/ para poder consultar la base de datos, pero se puede utilizar alguno que ya tengamos en nuestro ordenador.

Una vez instalados debemos seguir una serie de configuraciones que vienen bien explicadas en la página de UMLS http://www.nlm.nih.gov/research/umls/implementation_resources/scripts/README_ORF_MySQL_Output_Stream.html de las que resaltamos las más importantes:

- En el directorio donde tenemos instalado *MySql Server 5.5* hay un fichero de configuración *my.cnf* o *my.ini*, en el que debemos cambiar y añadir ciertos campos para optimizar la ejecución del programa indicados en la parte de configuración de parámetros de dicho enlace.
- En el directorio donde tenemos la base de datos 2013AB o la que tengamos en su caso, debemos modificar el fichero *populate_mysql_db.bat* en Windows o *populate_mysql_db.sh* en Linux. Debemos cambiar los campos de configuración con los nuestros. *set MYSQL_HOME=<path to MYSQL_HOME> set user=<username> set password=<password> set db_name=<db_name>*.

Una vez realizado los pasos anteriores debemos crear la base de datos que hemos llamado *umls* con una instrucción que introduciremos por consola en *mysqlserver5.5*.

```
CREATE DATABASE IF NOT EXISTS umls CHARACTER SET utf8 COLLATE
    utf8_unicode_ci;
```

Después de esto ya podemos ejecutar el *script populate_mysql_db.bat* que empezará a crear las tablas correspondientes. Debemos repetir el proceso para las tablas de la red semántica que están en la carpeta NET modificando *populate_net_mysql_db.bat* y ejecutándolo. Podemos ver el proceso en los ficheros *mysql.log* y *mysqlnet.log*. Una vez finalizados los *scripts* ya podemos trabajar con la base de datos en nuestras aplicaciones.

D.2. Algunas observaciones

- Se recomienda instalar *Mysql Community Server 5.5.37* como servicio de Windows y añadirlo como excepción al *fireware* de Windows para evitar cualquier problema con los puertos.
- Modificar el fichero *my.cnf* o *my.ini* con los valores recomendados:
  - key_buffer = 600M;
  - table_cache = 300;
  - sort_buffer_size = 500M;
  - read_buffer_size = 200M;
  - query_cache_limit = 3M;
  - query_cache_size = 100M;
  - myisam_sort_buffer_size = 200M;
  - bulk_insert_buffer_size = 100M;
  - join_buffer_size = 100M;
- Si alguno de los valores anteriores no es el adecuado (ver si nuestro ordenador es capaz de arrancar con esos valores) puede que no deje iniciar el servicio *MySql* ya que dará un error al intentar arrancarlo. Evidentemente si no se inicia correctamente el servicio, en nuestra aplicación saltarán excepciones relacionadas con la base de datos.
- Importante cambiar en el fichero de configuración del proyecto el nombre de la base de datos a la que hayamos puesto en la instrucción:

```
CREATE DATABASE IF NOT EXISTS umls CHARACTER SET utf8 COLLATE
    utf8_unicode_ci;
```

  Habrá excepciones si no se cambia adecuadamente. Estos ficheros se encuentran en el directorio *config* del proyecto y son *configGPS.xml* y *GPS.xml*[1].

[1] GPS: *Graph Processing System*.

- La base de datos que teníamos en el proyecto era `2008AB`. Era un subconjunto de `UMLS` que tenía principalmente las fuentes de `SNOMEDCT`. El problema de esto era que al ser un subconjunto muy reducido del total de información que tiene `UMLS` nos faltaban muchas relaciones y conceptos que no se encontraban en `SNOMEDCT`. Por lo que decidimos actualizar la base de datos a la `2013AB` e incluir la mayoría de fuentes de información para que no nos faltaran esas relaciones y conceptos. Un detalle importante es elegir el tamaño de ese subconjunto ya que cuanto más grande sea la base de datos más tiempo necesitará para hacer las consultas porque tiene que buscar mucha más información, lo que puede ralentizar la ejecución de la aplicación. En nuestro caso se generó una carpeta con 25GB aproximadamente y el tiempo de instalación ronda entre siete y doce horas. Es por ello importante que el programador elija los subconjuntos adecuados para su aplicación para intentar optimizar las consultas a la base de datos. Afortunadamente el paso del 2008 al 2013 no resultó gran problema debido a que la mayor parte de las tablas mantienen su nombre y las columnas iguales.

Apéndice E

Añadir plugins de layouts al proyecto

Los *layouts* que desarrolla la comunidad o que no están incluidos como parte de *Gephi*, se crean como *plugins*. Estos *plugins* pueden incluirse fácilmente en el entorno de *Gephi*, sin embargo no son compatibles con el *Toolkit* de *Gephi* si no se transforman previamente a librerías importables para *Java*. Para ello hay que usar una pequeña aplicación, `UnpackNBM`, que nos facilitará la tarea de convertir los *plugins* en archivos `*.jar`. Sólo hay que seguir los siguientes pasos:

1. Descargar `UnpackNBM`.
2. Abrir un terminal de línea de comandos e ir dónde hayamos descargado el programa. Después tenemos que escribir, especificando la ruta del archivo del *plugin* donde se indica:

   ```
   java -jar UnpackNBM.jar NBM-FILE-HERE
   ```

3. Los archivos `*.jar` aparecerán en la misma ruta que estaban los *plugins*.

Apéndice F

Glosario de términos

En el siguiente documento, agruparemos los términos distinguiendo entre herramientas y conceptos a los que nos referimos a lo largo de la memoria.

F.1. Conceptos

Acrónimos y abreviaturas: aparecen en multitud de textos de dominio técnico que sirven para abreviar una definición o extensión. *MetaMap* también aplica un algoritmo que trata de asociar acrónimos con expresiones.

Ambigüedad: uno de los problemas principales que afectan al PLN. Son situaciones en la que dos o más conceptos comparten un sinónimo.

GAR: generación Automática de Resúmenes.

Infoxication: sobrecarga de información.

Lexicón Especializado: disponible únicamente en inglés, es un diccionario que incluye términos y palabras inglesas de uso frecuente. Usados por herramientas léxicas para servir como ayuda en el PLN.

NegEx: algoritmo del que hace uso *MetaMap* para poder determinar cuando un concepto está negado.

NLM: *National Library of Medicine*.

Ontología: formulación de un exhaustivo y riguroso esquema conceptual dentro de uno o varios dominios dados, con la finalidad de facilitar la comunicación y el intercambio de información entre diferentes sistemas y entidades.

PLN: procesamiento del Lenguaje Natural. Consiste en un conjunto de técnicas de la rama de la Inteligencia Artificial cuyo objetivo es conseguir una comunicación hombre-máquina usando para ello el lenguaje con el que se comunican los hombres entre sí.

Red Semántica: puede usarse para clasificar vocabulario médico y se compone a su vez por los tipos semánticos y las relaciones semánticas.

Sistema experto: aplicación informática capaz de solucionar un conjunto de problemas que exigen un gran conocimiento sobre un determinado tema.

SNOMED-CT: *Systematized Nomenclature of Medicine Clinical Terms* es la terminología clínica integral, multi-lenguaje y codificada de mayor amplitud, precisión e importancia desarrollada en el mundo.

Tesauro: lista de palabras con significados similares, sinónimos, habitualmente acompañada por otra lista de antónimos. Normalmente está reducido a un campo concreto de la lengua como puede ser el ámbito biomédico.

UMLS: *Unified Medical Language System*, consiste en un conjunto de archivos y *software* que reúne gran cantidad de vocabularios biomédicos y de salud, y los estándares que dan la posibilidad de que distintos sistemas informáticos operen entre sí.

F.2. Herramientas

ANNIE: *A Nearly-New IE system*. Orientado a la extracción de información y análisis del lenguaje.

Gephi: herramienta libre para la visualización e interacción con grafos escrito en *Java*.

GATE: *General Architecture for Text Engineering,* `GATE` es una útil herramienta para el `PLN`.

MetaMap: software desarrollado por el *Dr. Alan Aronson* en la `NLM` de Estados Unidos que cuenta con múltiples opciones de configuración y permite mapear o asociar términos que aparecen en un texto biomédico, con conceptos del Metatesauro *UMLS*.

MetamorphoSys: herramienta multiplataforma que se actualiza en cada publicación de `UMLS`. Da la posibilidad de crear un subconjunto personalizado del Metatesauro.

Metatesauro: tesauro multi-lenguaje que contiene millones de conceptos biomédicos y sanitarios, sus nombres, sinónimos y sus relaciones utilizado en las diferentes herramientas de `UMLS`.

Bibliografía

[1] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[2] J. Barnes and P. Hut, "A hierarchical o (n log n) force-calculation algorithm," vol. 324, pp. 446–449, 1986.

[3] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[4] G. Consortium, "Gephi Tutorial Layouts," 2011. [Online]. Available: http://www.slideshare.net/gephi/gephi-tutorial-layouts

[5] C. Dangalchev, "Residual closeness in networks," *Physica A: Statistical Mechanics and its Applications*, vol. 365, no. 2, pp. 556–564, 2006.

[6] P. Dias, "Fromïnfoxication"toïnfosaturation": a theoretical overview of the cognitive and social effects of digital immersion," *Ambitos: Revista internacional de comunicación*, no. 24, pp. 31–40, 2014.

[7] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *J. Artif. Intell. Res.(JAIR)*, vol. 22, no. 1, pp. 457–479, 2004.

[8] I. C. R. Ferrer and R. Solé, "The small world of human language." in *Proceedings. Biological sciences/The Royal Society*, vol. 268, no. 1482, 2001, pp. 2261–2265. [Online]. Available: http://complex.upf.es/~ricard/SWPRS.pdf

[9] S. Fortunato and M. Barthelemy, "Resolution limit in community detection," *Proceedings of the National Academy of Sciences*, vol. 104, no. 1, pp. 36–41, 2007. [Online]. Available: http://arxiv.org/abs/physics/0607100

[10] L. Hervás Martín, V. Martínez Simón, and I. Sánchez Martínez, "Extracción de información en informes médicos," 2013, proyecto de Sistemas Informáticos (Facultad de Informática, Curso 2012-2013). [Online]. Available: http://eprints.ucm.es/22563/

[11] S. M. Humphrey, W. J. Rogers, H. Kilicoglu, D. Demner-Fushman, and T. C. Rindflesch, "Word sense disambiguation by selecting the best semantic type based on journal descriptor indexing: Preliminary experiment," *Journal of the American Society for Information Science and Technology*, vol. 57, no. 1, pp. 96–113, 2006. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2771948/

[12] R. Lambiotte, J.-C. Delvenne, and M. Barahona, "Laplacian dynamics and multiscale modular structure in networks," *arXiv preprint arXiv:0812.1770*, 2008. [Online]. Available: http://arxiv.org/abs/0812.1770

[13] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146. [Online]. Available: http://kowshik.github.io/JPregel/pregel_paper.pdf

[14] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006. [Online]. Available: http://arxiv.org/abs/physics/0602124

[15] J. Nielsen, "Im, not ip (information pollution)," *Queue*, vol. 1, no. 8, pp. 76–75, 2003. [Online]. Available: http://www.nngroup.com/articles/information-pollution/

[16] L. Plaza, "Uso de grafos semánticos en la generación automática de resúmenes y estudio de su aplicación en distintos dominios: Biomedicina, periodismo y turismo," *Periodismo y Turismo*, 2011, tesis Doctoral. [Online]. Available: http://eprints.ucm.es/12662/

[17] J. Sun and J. Tang, "A survey of models and algorithms for social influence analysis," in *Social Network Data Analytics*. Springer, 2011, pp. 177–214.

[18] Wikipedia, "Centrality — Wikipedia, The Free Encyclopedia," 2014. [Online]. Available: http://en.wikipedia.org/wiki/Centrality

[19] Wikipedia, "Barnes-Hut simulation — Wikipedia, The Free Encyclopedia," 2014. [Online]. Available: http://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation

[20] Wikipedia, "Graph drawing — Wikipedia, The Free Encyclopedia," 2014. [Online]. Available: http://en.wikipedia.org/wiki/Graph_drawing#Layout_methods

[21] Wikipedia, "Scale-free network — Wikipedia, The Free Encyclopedia," 2014. [Online]. Available: http://en.wikipedia.org/wiki/Scale-free_network

[22] Wikipedia, "Ontologia(informatica) — Wikipedia, The Free Encyclopedia," 2014. [Online]. Available: http://es.wikipedia.org/wiki/Ontolog%C3%ADa_(inform%C3%A1tica)

[23] I. Yoo, X. Hu, and I.-Y. Song, "A coherent graph-based semantic clustering and summarization approach for biomedical literature and a new summarization evaluation method," *BMC bioinformatics*, vol. 8, no. Suppl 9, p. S4, 2007.