Herramienta para Filtrado Estructural sobre Ontología Gene Ontology.

Tesina de grado presentada
por

Antonela Mariel Destito
D-2271/3

al
Departamento de Ciencias de la Computación
en cumplimiento parcial de los requerimientos
para la obtención del grado de

Licenciada en Ciencias de la Computación

Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario
Av. Pellegrini 250, Rosario, República Argentina

7 de diciembre de 2013

Directora

Dra. Pilar Bulacio

Cifasis
27 de Febrero 210bis
Rosario, Argentina

“Entre las dificultades
se esconde la oportunidad”

Albert Einstein

Resumen

La *bioinformática* es la aplicación de tecnologías computacionales a la gestión y al análisis de datos biológicos. En la actualidad, los datos recolectados de los experimentos biológicos son guardados en grandes bases de datos. Estos datos están relacionados con un dominio del problema, en el cual es necesario definir los conceptos principales y las relaciones entre ellos. La organización de dichos conceptos se realiza mediante el diseño de ontologías. La herramienta OBO-Edit es una de las más usadas para la visualización y edición de ontologías. Su desarrollo en Berkeley Informatics a través de Gene Ontology Consortium consideró las necesidades de diseño dentro del ámbito biológico: complejidad de los datos, necesidades simples de razonamiento rápido, potencia de búsqueda, posibilidad de filtrado. Una de las funcionalidades principales que posee OBO-Edit es poder obtener subontologías para aquellos casos en donde se necesiten realizar estudios exhautivos sólo en partes de las ontologías.

En este trabajo nos centramos en el filtrado como el proceso para hacer posible la división del complejo problema biológico en subproblemas. Debemos notar que el proceso de filtrado dentro de los problemas biológicos implica considerar por un lado el filtrado de conceptos (filtrado sobre el esquema ontológico) y por el otro, el filtrado de los datos asociados a dichos conceptos (filtrado en bases de datos). Los especialistas son los encargados de seleccionar las subontologías con las que trabajar y filtrar las bases de datos para obtener los datos que se corresponden con dichas subontologías. Este proceso de filtrado es complejo y requiere de etapas manuales. Una etapa requiere filtrar las ontologías y, otra, las bases de datos. Este problema nos motiva a poder brindar una solución a la simplificación y automatización del proceso de filtrado, haciendo más sencilla la tarea de los biólogos. Con este fin, proponemos la automatización de estos dos procesos de filtrado en un proceso único y transparente que permita manipular de forma simple tanto las ontologías como las bases de datos.

Índice

1. Motivación y Objetivo General **9**

2. Introducción **12**
2.1. Gene Ontology 12
2.1.1. La Estructura Ontológica 13
2.1.1.1. Componente Celular 14
2.1.1.2. Proceso Biológico 14
2.1.1.3. Función Molecular 14
2.1.1.4. Relaciones en la Ontología 14
2.1.2. Proceso de Anotación 15
2.2. Herramientas 15
2.2.1. AmiGO 15
2.2.2. GOOSE 16
2.2.3. OBO-Edit 16
2.2.4. QuickGO 17
2.3. Discusión 17

3. Objetivos Específicos **18**

4. Diseño **19**
4.1. Diseño del Filtro de Ontologías 19
4.2. Diseño del Filtro de Bases de Datos 21
4.3. Diseño del Pipeline de Filtrado 22
4.3.1. Entrada 23
4.3.2. Proceso de Filtrado 28
4.3.3. Salida 29
4.3.4. Sintonizadores 29
4.4. Discusión 30

5. Especificación **31**
5.1. Caracterización de Actores 31
5.2. Casos de Uso 32
5.3. Discusión 34

6. Aspectos de Implementación **35**
6.1. Filtro 1: Filtro de Ontologías 35
6.1.1. Filtro MF 37
6.2. Filtro 2: Filtro de Bases de Datos 38
6.2.1. Filtro DB 40
6.3. Confección de GUIs 40
6.3.1. Interfaz Gráfica 1: Filtro de Ontologías 41
6.3.2. Interfaz Gráfica 2: Filtro de Bases de Datos 41

6.3.3. Interfaz Gráfica 3: Pipeline 42
6.4. Discusión 43

7. Conclusión y Trabajos Futuros **44**
7.1. Trabajos Futuros 45

8. Apéndices **47**
8.1. Apéndice A: Tecnologías Utilizadas 47
8.2. Apéndice B: Herramientas de Desarrollo 49
8.3. Apéndice C: Códigos Fuente de OBO-Edit 52
8.4. Apéndice D: Filtro MF Manual 53
8.5. Apéndice E: Modificaciones Realizadas al Proyecto OBO-Edit 57

Bibliografía **83**

Glosario **85**

Índice de figuras

1. Representación esquemática de la rama Biological Process de GO con OBO-Edit. Los recuadros representan conjuntos de conceptos. 13
2. Filtrado de estructuras ontológicas. 20
3. Filtrado de bases de datos. 22
4. Pipeline de filtrado de estructura ontológica GO y obtención de instancias asociadas a una DB siguiendo la Arquitectura de Software "Tubos y Filtros". 23
5. Ontología editada con OBO-Edit. 24
6. Fragmento de la anotación *Arabidopsis thaliana*. 25
7. Información del término GO:0000015. 27
8. Árbol de inferencia del término GO:0000015. 27
9. Ontología filtrada por la función molecular editada con OBO-Edit. 28
10. Fragmento de la anotación de *Arabidopsis thaliana* luego de aplicarle el Filtro 2 con opciones IDA e ISS. 29
11. Actores y relaciones entre ellos. 31
12. Diseño del Filtro de Ontologías. 36
13. Diseño del Filtro de Bases de Datos. 39
14. Interfaz Gráfica para el Filtro de Ontologías. 41
15. Interfaz Gráfica para el Filtro de Bases de Datos. 42
16. Interfaz Gráfica para el Pipeline de Filtrado. 43
17. Un diagrama sobre el proceso de desarrollo de software. . . . 47
18. La API y la JVM aíslan al programa del hardware subyacente. 47
19. Vista de las dos solapas: Source y Design. 50
20. Vista de la solapa Design y las herramientas de diseño de interfaz gráfica. 51
21. Vista de OBO-Edit al cargar una ontología. 53
22. Vista de OBO-Edit con las opciones de filtrado. 54
23. Object Filtering en OBO-Edit. 55
24. Link Filtering en OBO-Edit. 55
25. Tag Filtering en OBO-Edit. 56

Organización del Trabajo

Capítulo 1: Motivación y Objetivo General

Presentamos en este capítulo la motivación que nos llevó a conformar el siguiente trabajo y cuáles son los objetivos generales a cumplir.

Capítulo 2: Introducción

En este capítulo describimos el contexto del trabajo. Cuenta de varias secciones referidas a la descripción de ontologías GO y a las herramientas para manipular las ontologías y las bases de datos.

Capítulo 3: Objetivos Específicos

Mostramos aquí los objetivos específicos del presente trabajo, los cuales derivan del objetivo general planteado en el Capítulo 1.

Capítulo 4: Diseño

Este capítulo abarca el desarrollo de los objetivos específicos. Definimos el diseño de los distintos filtros. Hacemos hincapié en la arquitectura de filtrado para posterior implementación del *pipeline*.

Capítulo 5: Especificación

Especificamos los distintos filtros a través de los actores y los casos de uso.

Capítulo 6: Aspectos de Implementación

Indicamos cómo se lleva a cabo la implementación de los procesos de filtrado y del *pipeline*. Presentamos el diseño orientado a objetos utilizado, además de un detalle de las clases y los métodos utilizados o creados.

Capítulo 7: Conclusión y Trabajos Futuros

Finalizamos el trabajo, mostrando las conclusiones obtenidas. Presentamos ideas que pueden ser consideradas para la extensión de esta tesina.

Apéndices

Con el propósito de poder recrear los pasos realizados en este trabajo, mostramos detalles técnicos sobre los programas y códigos fuente utilizados. Además añadimos en un último apéndice los códigos correspondientes a las clases modificadas/agregadas al proyecto.

En el Apéndice A mostramos aspectos del lenguaje de programación Java relevantes a la implementación de la herramienta de diseño de ontologías OBO-Edit. En el Apéndice B presentamos el entorno de desarrollo Eclipse, plugins de Eclipse para Edición Gráfica y el repositorio de versiones SVN, útiles para poder manipular el proyecto OBO-Edit. En el Apéndice C indicamos los pasos a seguir para la obtención de los códigos fuente del proyecto. En el Apéndice D mostramos cómo realizar el filtro *molecular function* sobre la ontología con OBO-Edit de forma manual. El Apéndice E contiene los códigos fuente que se modificaron y/o agregaron al proyecto.

1. Motivación y Objetivo General

La *bioinformática*[1] es la aplicación de la ciencia y la tecnología de la computación a la gestión y análisis de datos biológicos. El término bioinformática hace referencia a campos de estudios *interdisciplinarios* que requieren conceptos de diferentes áreas que incluyen informática (Bajic, Brusic, Li, kiong Ng, y Wong, 2003), matemática aplicada (Lander, Michael S. Waterman, y Protein Structure Research, 1995), estadística (Woon, 2003), ciencias de la computación (Crandall y Lagergren, 2008), inteligencia artificial (Frasconi y Shamir, 2003), química (Murray-Rust, Mitchell, y Rzepa, 2005) y bioquímica (Ibba, 2002) para analizar datos, simular sistemas, todos ellos de índole biológico, y usualmente en el nivel molecular (Altman, 2006). El núcleo principal de la bioinformática se encuentra en la utilización de recursos computacionales tales como algoritmos, bases de datos, tecnología web, soft computing[2], minería de datos, teoría de sistemas y control, estadística, entre otros, para generar nuevos conocimientos biológicos sobre modelos informáticos[3]. Parte de los principales esfuerzos de investigación en estos campos incluyen el alineamiento de secuencias, la predicción de genes, montaje del genoma, alineamiento estructural de proteínas, predicción de estructura de proteínas, predicción de la expresión génica, interacciones proteína-proteína, y modelado de la evolución (Kanehisa y Bork, 2003).

La interdisciplinariedad de la bioinformática hace indispensable convenir una descripción común de los conceptos y la relación entre ellos. Con este fin se proponen diversas ontologías[4]. Una de las más relevantes es Gene Ontology (GO)[5], sobre la cual nos enfocaremos. GO está estructurada[6] como un grafo acíclico dirigido. Cada término tiene definidas las relaciones *is a*, *part of* y *regulates*. Los conceptos descriptos por GO se dividen en tres grandes ramas: *(i)* Componente Celular, que describe las partes de una célula o su entorno extracelular; *(ii)* Función Molecular, que describe las actividades

[1] Definición dada por European Bioinformatics Institute. `http://www.ebi.ac.uk/2can/bioinformatics/bioinf_what_1.html`, Fecha de consulta: Agosto 2012.

[2] *Soft computing* ofrece métodos de búsqueda y de razonamiento para resolver problemas con información incompleta, incertidumbre, y/o inexactitud. Para ello utiliza técnicas de lógica borrosa, probabilidad, redes neuronales y algoritmos genéticos. (Bonissone, 1997; Zadeh, 1994).

[3] Definición extraída de la página de la Universidad de Alabama en Birmingham, Escuela de Medicina. `http://hydrogen.path.uab.edu/bioinformatics.html`, Fecha de consulta: Agosto 2012.

[4] El término *ontología* en informática hace referencia a la formulación de un exhaustivo esquema conceptual dentro de un dominio, con la finalidad de posibilitar el intercambio de información entre diferentes sistemas o entidades. `http://www.csd.com.es/FOG/index.html?seccion=inicio`, Fecha de consulta: Agosto 2012.

[5] El proyecto Gene Ontology provee un vocabulario controlado que describe el gen y los atributos del producto génico en cualquier organismo.

[6] Para más detalle de la estructura ontológica de GO: `http://www.geneontology.org/GO.ontology.structure.shtml`, Fecha de consulta: Agosto 2013.

elementales de un material genético a nivel molecular; *(iii)* Proceso Biológico, que describe las operaciones moleculares asignadas a la funcionalidad de unidades integradas de vida (células, órganos, organismos, etc.).

GO se utiliza para estructurar conceptualmente los resultados de los experimentos biológicos de manera formal, siguiendo el vocabulario estructurado de la ontología. El gran volumen de datos asociados a dichos resultados es organizado en bases de datos (DBs) de millones de entradas. Debemos notar que desde el punto de vista biológico, el desarrollo de las ontologías posibilitó el análisis de una inmensa cantidad de información y la obtención de datos relevantes como por ejemplo, las anotaciones. Una *anotación* es la declaración de una conexión entre una muestra de producto genético y los términos de la ontología.

Un problema frecuente que deben resolver los biólogos es intentar anotar un conjunto de muestras de datos genómicos que obtuvieron a partir de sus experimentos. Una primera forma para lograr este objetivo es plantear nuevos ensayos para verificar que la muestra está asociada a un término GO en particular. Una segunda forma es predecir la anotación a través de un método computacional basado en aprendizaje automatizado. Este último caso, denominado anotación electrónica (IEA: Inferred from Electronic Annotation), es donde contribuiremos con esta tesina. Para realizar el proceso de predicción se suele diseñar un esquema de clasificadores supervisados capaces de tipificar cada muestra como perteneciente o no a cada término GO. Para generar los conjuntos de datos de entrenamiento de cada clasificador es necesario contar con un conjunto de muestras significativas asociadas al término GO del clasificador. Bajo este objetivo se puede notar que un paso necesario es filtrar las anotaciones de las grandes bases de datos seleccionando diversos organismos y los diferentes códigos de evidencia[7] de anotación a utilizar.

Otro problema habitual, es que la gran dimensión conceptual del dominio biológico hace inevitable que los biólogos se especialicen sobre una parte de la ontología, es decir, sobre subestructuras o subgrafos requiriendo también el filtrado de la estructura de la ontología con sus conceptos y relaciones, junto con las instancias de alguna base de datos asociadas a dichos conceptos. Debemos notar que aunque este proceso de filtrado se requiere en forma habitual, no hay una herramienta que lo realice en forma automatizada, sino que se hace en diferentes etapas configuradas manualmente junto con la ejecución de diferentes *scripts*. Este proceder no sistemático hace que el proceso sea complejo, no intuitivo y poco robusto, lo cual nos motiva al objetivo general de la presente tesina:

Diseñar e implementar una herramienta computacional para gestionar el

[7]El código de evidencia GO refleja el protocolo seguido en el proceso de anotación. Para más información consultar: `http://www.geneontology.org/GO.evidence.shtml`, Fecha de consulta: Agosto 2013

filtrado sistemático, flexible y transparente sobre la ontología GO, obteniendo la subestructura conceptual jerárquica y los datos asociados (anotaciones) a dicha estructura en las bases de datos correspondientes.

Donde "flexible" implica que el proceso sea susceptible a cambios según un conjunto de opciones relevantes para el ámbito biológico, factibles de ser variadas para obtener un filtro personalizado; y "transparente" implica que el usuario no percibe los procesos internos.

Para ello se diseñará una arquitectura tipo *pipeline* que reutilizará parte de las herramientas actuales en forma organizada modificando y/o desarrollando el software necesario para lograr un proceso flexible y transparente de filtrado. Las entradas del *pipeline* serán una ontología y una base de datos, y como salida se deberán obtener los datos asociados a la ontología filtrada.

En lo que sigue de esta tesina empezamos por introducir aspectos importantes del proyecto GO y mostramos las herramientas para poder editar/manipular las ontologías, enfocándonos en lo que respecta al proceso de filtrado, describiendo sus ventajas y desventajas. Luego formalizamos los objetivos específicos a seguir, y esbozamos cómo a través de un *pipeline* de filtrado serán logrados.

2. Introducción

A modo de introducción, presentamos brevemente algunos aspectos del proyecto Gene Ontology, abarcando su estructura y sus relaciones. También mostramos uno de sus principales usos que es el proceso de anotación, que sirve para vincular términos GO a los datos. Por último, detallamos las herramientas para poder manipular las ontologías, entre ellas OBO-Edit.

2.1. Gene Ontology

El proyecto GO es un esfuerzo colaborativo que logra describir productos genéticos y sus relaciones. Comenzó en 1998 como una colaboración entre investigadores de tres bases de datos de organismos modelo[8]: FlyBase[9] (Drosophila), Saccharomyces Genome Database[10] (SGD) y Mouse Genome Database[11] (MGD) dando lugar al GO Consortium[12]. Desde ese momento diversos genomas de plantas, animales y microbios han sido incluídos. Para poder lograr sus objetivos de descripción formal de conceptos asociados al dominio de la biología molecular, el proyecto GO se encarga de las siguientes tareas:

(i) El mantenimiento y desarrollo de un vocabulario controlado que describe atributos de los genes y productos genéticos.

(ii) Las anotaciones de los productos genéticos a través de la asignación de términos GO, que implican realizar asociaciones entre las ontologías, los genes y productos genéticos en las DBs.

(iii) El desarrollo de herramientas que faciliten la creación, mantenimiento y utilización de las ontologías y las anotaciones.

El uso de anotaciones hace posible, en primer lugar, vincular conceptos descriptivos a datos biológicos. En segundo lugar, realizar consultas relativas a conceptos en las DBs encontrando analogía en el comportamiento de diversos organismos. Por ejemplo, se puede usar GO para encontrar todos los

[8] Un organismo modelo es un organismo biológico representativo de un taxón (grupo de organismos emparentados), útil para inferir fenómenos biológicos. Por ejemplo, Arabidopsis thaliana en plantas; Escherichia coli en bacterias; Mus musculus para humanos.

[9] Para más información sobre la base de datos FlyBase: `http://flybase.org/`, Fecha de consulta: Agosto 2013

[10] Para más información sobre la base de datos Saccharomyces Genome Database: `http://www.yeastgenome.org/`, Fecha de consulta: Agosto 2013

[11] Para más información sobre la base de datos Mouse Genome Database: `http://www.informatics.jax.org/`, Fecha de consulta: Agosto 2013

[12] El *GO Consortium* es un conjunto de modelos de organismos, bases de datos de proteínas y comunidades de búsqueda biológica involucrados en el desarrollo de aplicaciones para GO. `http://www.geneontology.org/GO.consortiumlist.shtml`, Fecha de consulta: Agosto 2013

productos genéticos involucrados en transducción[13] de señales en el genoma de un ratón, o examinar los receptores de la enzima *tyrosine kinases* en varios organismos. La estructura de GO también permite a los anotadores que asignen propiedades a los genes o productos genéticos a diferentes niveles de detalle, dependiendo del nivel de conocimiento de la entidad analizada. Notemos que su estructura de grafo permite a través de la relación *is a* definir distintos niveles de generalización o especificación de clases (conceptos).

El proyecto GO ha desarrollado tres ramas o estructuras ontológicas[14] que describen a los productos genéticos en términos, según sus procesos biológicos, sus componentes celulares y sus funciones moleculares. A continuación describimos algunos aspectos de las estructuras ontológicas.

2.1.1. La Estructura Ontológica

GO es un vocabulario controlado, un conjunto de conceptos y relaciones para describir un dominio de información. Con este fin se definen términos y las relaciones posibles entre ellos, logrando así un vocabulario controlado.

La estructura de la ontología es un grafo acíclico dirigido donde cada término o concepto es un nodo y las relaciones entre los términos son los arcos del grafo. El grafo dirigido puede definir una jerarquía entre nodos padres y nodos hijos (clases generalizadoras y específicas). La Fig. 1 muestra una de las tres ramas descriptas por GO, *biological process*:

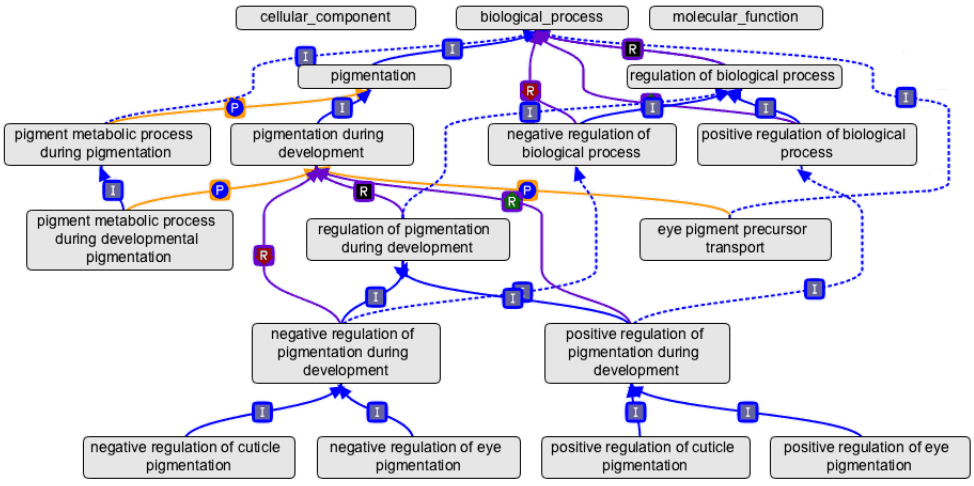

Figura 1: Representación esquemática de la rama Biological Process de GO con OBO-Edit. Los recuadros representan conjuntos de conceptos.

[13] Transformación de un tipo de señal en otro distinto.

[14] Las ontologías pueden descargarse desde: `http://www.geneontology.org/GO.downloads.ontology.shtml`, Fecha de consulta: Agosto 2013

2.1.1.1. Componente Celular Se denomina componente celular a un elemento de una célula, una parte de un objeto, el cual puede ser una estructura anatómica (por ejemplo, retículo endoplasmático rugoso o núcleo) o un grupo de productos genéticos (por ejemplo, ribosoma, proteasoma o una proteína dimérica).

2.1.1.2. Proceso Biológico Un proceso biológico es una serie de eventos o funciones moleculares. Algunos ejemplos de términos de proceso biológico son: *cellular physiological process*, *signal transduction*, *pyrimidine metabolic process*, *alpha-glucoside transport*.

Puede ser difícil distinguir entre proceso biológico y función molecular, pero la regla general es que un proceso se debe realizar en más de un paso de funciones moleculares.

2.1.1.3. Función Molecular La función molecular describe actividades que ocurren en el nivel molecular. Los términos GO para función molecular representan las actividades que las entidades llevan a cabo, sin especificar dónde, cuándo o en qué contexto se realizan. Las actividades que describen las funciones moleculares generalmente son realizadas por productos genéticos individuales, pero algunas son realizadas por productos genéticos complejos. Algunos términos de función molecular son: *catalytic activity*, *transporter activity*, *binding*, *adenylate cyclase activity*, *toll receptor binding*.

Es fácil confundir el producto genético con su función molecular, para ello, las funciones moleculares GO llevan la palabra *activity*.

2.1.1.4. Relaciones en la Ontología Las relaciones son los arcos de la ontología vista como grafo. Describimos a continuación las tres relaciones más importantes de GO.

- is a

  A *is a* B, significa que A es un subtipo del nodo B.

- part of

  A *part of* B, significa que A es necesariamente parte de B: si A existe, tiene una parte de B y la presencia de A implica la presencia de B.

- regulates

  A *regulates* B, significa que el proceso A afecta directamente la manifestación del proceso B.

2.1.2. Proceso de Anotación

Las anotaciones[15] surgen con el propósito de vincular a los datos biológicos con los términos GO, los cuales nombran a los conceptos biológicos dentro de la ontología. Para realizar las anotaciones se capturan las actividades y la localización de un producto genético en uno o varios términos GO. Los miembros de GO Consortium hacen sus propias anotaciones que están disponibles al público, formando parte de los datos que se acceden desde el navegador GO o desde sus herramientas.

Podemos notar, debido a la magnitud del proyecto, que existen versiones reducidas de GO, como por ejemplo, las *GO slims*[16]. Las versiones simplificadas son útiles para anotar genomas o conjuntos de productos genéticos. Por ejemplo, se puede averiguar la proporción de un genoma involucrado en transducción de señales, biosíntesis o reproducción. Las GO slims las pueden crear los usuarios de acuerdo a sus necesidades, como especies específicas o áreas particulares de las ontologías.

2.2. Herramientas

Dentro de esta sección mostramos algunas de las principales herramientas[17] existentes para el filtrado de estructuras ontológicas GO junto con los datos asociados a dichas estructuras. Describiremos brevemente AmiGO, GOOSE, OBO-Edit (desarrolladas por el GO Consortium) y QuickGO (desarrollada por UnitProt-GOA). Estas herramientas han sido seleccionadas por ser las más populares dentro del ámbito de los biólogos para realizar tareas de edición y procesamiento de ontologías y anotaciones.

2.2.1. AmiGO

AmiGO[18] provee una interfaz de búsqueda y navegación de ontologías y anotaciones. La herramienta ofrece diferentes formas de manejo de las ontologías. En cuanto al filtrado, se pueden realizar filtros[19] predefinidos de la ontología GO según sus estructuras ontológicas (proceso biológico, componente celular o función molecular), eligiendo las anotaciones de las cuales se precisa extraer los datos. Estos filtros existentes resultan limitados ya que el usuario no cuenta con un conjunto de opciones para armar un filtro

[15] Las anotaciones pueden descargarse desde: `http://www.geneontology.org/GO.current.annotations.shtml`, Fecha de consulta: Agosto 2013

[16] Las GO slims pueden descargarse desde: `http://www.geneontology.org/GO.slims.shtml`, Fecha de consulta: Agosto 2013

[17] Para más detalle de las herramientas de GO: `http://www.geneontology.org/GO.tools.shtml`, Fecha de consulta: Agosto 2013

[18] Para realizar búsquedas con AmiGO: `http://amigo.geneontology.org/cgi-bin/amigo/go.cgi`, Fecha de consulta: Agosto 2013

[19] Para realizar filtros con AmiGO: `http://amigo.geneontology.org/cgi-bin/amigo/browse.cgi?session_id=6294amigo1338470403`, Fecha de consulta: Agosto 2013

personalizado. Tampoco es posible ingresar una ontología o base de datos para filtrar, sino que se deben utilizar las ontologías y las bases de datos que ya están cargadas en la herramienta, impidiendo en algunos casos trabajar con los datos deseados.

Otra posibilidad que brinda AmiGO es hacer búsquedas de términos, productos genéticos o proteínas en las DBs de GO a través de consultas[20] SQL, lo cual es dificultoso para el uso entre biólogos ya que demanda más conocimiento informático.

2.2.2. GOOSE

Otra herramienta útil para filtrado de DBs es la llamada GOOSE[21], que es un ambiente SQL para consulta sobre GO.

Nuevamente brinda gran flexibilidad y rapidez debido al filtrado directo sobre DBs[22], sin embargo, SQL es un lenguaje complejo para los biólogos. Además se puede perder de vista la estructura ontológica subyacente, ya que sólo se trabaja sobre las DBs. Tampoco es posible ingresar bases de datos de forma manual.

2.2.3. OBO-Edit

OBO-Edit[23] es un editor de ontologías que permite crearlas, modificarlas y realizar filtrados ya sea en función de los términos GO como de las relaciones definidas entre términos. Un ejemplo de filtro de términos es aquel que permite encontrar todos los términos de la ontología que tengan la palabra *kinase* en su nombre. Un ejemplo de filtro de relaciones extrae los vínculos de la ontología relacionados por *part of*. Sin pérdida de generalidad, uno de los filtros complejos donde centraremos nuestro trabajo es el filtro *molecular function*[24] (MF), necesario para los biólogos en tareas de predicción automática de funcionalidades de proteínas (Lu y Hunter, 2005). Notemos que las soluciones de diseño de este filtro pueden extenderse fácilmente a otros filtros en otras ramas de GO.

Si bien OBO-Edit es totalmente flexible y completo a la hora de armar filtros sobre las ontologías, sólo logra parte de la tarea de filtrado de GO: obtiene la subestructura ontológica pero no los datos de alguna DB asociados a esa subestructura. Para ello, necesitaremos utilidades adicionales. Por otro

[20] Para realizar consultas con AmiGO: `http://amigo.geneontology.org/cgi-bin/amigo/blast.cgi?session_id=7332amigo1338473328`, Fecha de consulta: Agosto 2013

[21] Para realizar consultas SQL con la herramienta GOOSE: `http://www.berkeleybop.org/goose/`, Fecha de consulta: Agosto 2013

[22] Para más información sobre el esquema de DBs de GO: `http://www.geneontology.org/GO.database.schema.shtml`, Fecha de consulta: Agosto 2013

[23] Para más información sobre el editor OBO-Edit y para realizar la descarga del programa: `http://www.oboedit.org/`, Fecha de consulta: Agosto 2013

[24] Para más información sobre cómo realizar el filtro Molecular Function en OBO-Edit ver el Apéndice D (8.4)

lado, el armado de las opciones de filtrado suele ser complicado, dado que hay que seguir varios pasos para configurar un filtro. En el Apéndice D (8.4) mostramos información adicional de cómo lograr el filtro MF de forma manual.

2.2.4. QuickGO

QuickGO[25] es un rápido buscador vía Web para términos y anotaciones GO, desarrollado por el grupo UnitProt-GOA[26] del European Bioinformatics Institute. Posee un buscador con filtrado de anotaciones GO que permite obtener términos y datos asociados. No obstante, debemos notar que permite aplicar un pequeño grupo de parámetros fijos de filtrado con el fin de obtener conjuntos de anotaciones y el buscador recolecta los datos restringido sólo a la base de datos UniProtKB[27].

Al no poseer flexibilidad en el armado de filtros o en la elección de la base de datos a utilizar, la herramienta resulta limitada.

2.3. Discusión

Luego de una exploración sobre las herramientas comúnmente usadas en el dominio de problemas genómicos para el diseño y/o gestión de estructuras ontológicas junto con sus datos asociados, pudimos observar que no existe una herramienta que brinde la posibilidad de realizar un filtrado flexible sobre la ontología y sobre sus instancias asociadas. Podemos mencionar que existen aquellas que están avocadas al diseño y gestión de las estructuras ontológicas, otras que posibilitan realizar filtrados *duros* sobre sus bases de datos, y por último, las que realizan filtrados *predefinidos* tanto de la estructura ontológica como de sus datos. Esta limitación de tareas realizables por cada una de las herramientas tiene por objetivo acotar la complejidad computacional: la descripción o exploración de conceptos GO del dominio genómico es enorme, y la generación o búsqueda de instancias asociadas a esos conceptos también lo es. Todo lo anteriormente explicitado nos motiva a plantear el objetivo de la presente tesina destinado a generar un filtrado completo y flexible sobre GO y sus DBs asociadas, integrando consistentemente las herramientas existentes.

En el siguiente capítulo presentamos los objetivos específicos de la tesina, los cuales desarrollamos exhaustivamente en los capítulos posteriores.

[25] Para realizar una búsqueda con QuickGO: `http://www.ebi.ac.uk/QuickGO/`, Fecha de consulta: Agosto 2013

[26] Para más información sobre el grupo Uniprot-GOA y sus anotaciones: `http://www.ebi.ac.uk/GOA/uniprot_release.html`, Fecha de consulta: Agosto 2012

[27] Para una información detallada de UniProtKB: `http://www.uniprot.org/help/uniprotkb`, Fecha de consulta: Agosto 2013

3. Objetivos Específicos

Considerando el objetivo propuesto: *Diseñar e implementar una herramienta computacional para gestionar el filtrado sistemático, flexible y transparente sobre la ontología GO, obteniendo la subestructura conceptual jerárquica y los datos asociados (anotaciones) a dicha estructura en las bases de datos correspondientes*, proponemos el diseño de una arquitectura tipo *pipeline* que reutilizará parte de las herramientas actuales en forma organizada modificando y/o desarrollando el software necesario para lograr un proceso flexible y transparente de filtrado. Para ello, presentamos los siguientes objetivos específicos:

(i) Exploración y selección de herramientas existentes para la manipulación de ontologías GO y DBs;

(ii) Diseño de un filtro automático sobre la ontología GO que posibilite obtener una subestructura ontológica;

(iii) Diseño de un filtro que recupere en forma automática las instancias de las DBs asociadas a la subestructura obtenida en (ii);

(iv) Integración de ambas etapas en un único *pipeline* que posibilite un proceso de filtrado transparente para el biólogo quien deberá fijar las condiciones de entrada para obtener la subestructura y las instancias como salida.

Estos objetivos serán desarrollados dentro del editor de ontologías OBO-Edit puesto que es una herramienta de código abierto diseñada por el Consorcio GO. Sin embargo, debemos notar que los resultados obtenidos podrían extenderse a otros editores de código abierto, por ejemplo, Protégé[28].

En los próximos capítulos nos abocamos al desarrollo particular de cada uno de los objetivos específicos expuestos anteriormente. Detallamos cómo se lleva a cabo cada objetivo, mostrando caminos a seguir, complicaciones encontradas, diseños y especificaciones.

[28] Sitio web del editor: `http://protege.stanford.edu/`, Fecha de consulta: Diciembre 2013

4. Diseño

En este capítulo detallamos las etapas llevadas a cabo para alcanzar nuestro objetivo de diseño de herramienta de filtrado flexible de estructuras ontológicas GO junto con sus instancias asociadas. Para ello, realizamos la descripción de nuestro trabajo en torno a los objetivos específicos planteados en el capítulo anterior.

En el capítulo de Introducción (Capítulo 2) se realizó un estudio minucioso de las herramientas frecuentemente utilizadas dentro de dominios de problemas genómicos en el manejo y/o diseño, tanto de estructuras ontológicas como en las estructuras de guardado de datos (instancias de conceptos) asociados. A partir de este estudio vimos que debido a la complejidad del problema, en general las herramientas realizan o el filtrado de las estructuras ontológicas o el filtrado de los datos en las DBs. Asimismo, existen unas pocas herramientas que realizan el filtrado completo de estructura ontológica junto con sus instancias asociadas a partir de filtros sencillos y predefinidos.

En consecuencia, considerando la naturaleza del problema y las herramientas existentes, plateamos la tarea de filtrado completo y flexible en dos etapas conexas: *i*) el filtrado de la estructura ontológica GO, y *ii*) el filtrado consistente en una DB de las instancias asociadas a la subestructura obtenida en *i*). Debemos notar que la conexión transparente de estas dos etapas se realiza mediante una integración a partir de una arquitectura de procesamiento tipo *pipeline*, cuyos bloques son formulados en el transcurso del presente capítulo.

4.1. Diseño del Filtro de Ontologías

Partiendo del análisis y exploración de herramientas que realizan el filtrado de ontologías, hemos visto que AmiGO o QuickGO proveen sólo filtros predefinidos de ontologías. Por otro lado, la herramienta OBO-Edit implementada por el consorcio GO para el diseño y edición de ontologías, permite además la configuración de filtros considerando una gran cantidad de opciones de filtrado de estructuras ontológicas definidas en los archivos `.obo`, pero no considera la obtención de instancias asociadas a la subestructura. Debemos notar, sin embargo, que la flexibilidad dada por la cantidad de alternativas a seleccionar en OBO-Edit hace que el armado del filtro no sea trivial. Por lo tanto, dentro de los requerimientos en esta etapa facilitaremos dentro de OBO-Edit la posibilidad de usar filtros automáticos generados a partir de las opciones intuitivas empleadas por los biólogos.

Teniendo en cuenta lo expuesto anteriormente sugerimos los siguientes requerimientos para el filtro de estructuras ontológicas en OBO-Edit:

(i) Diseñar opciones de filtrado automático para filtrar ontologías ingresadas por el usuario.

(ii) Realizar un filtrado transparente de la ontología según la opción elegida reutilizando las funciones de filtrado ya existentes en la herramienta OBO-Edit.

(iii) Retornar un archivo de salida que contenga los términos de la estructura ontológica filtrada (subestructura) según el filtro seleccionado previamente por el usuario.

La Fig. 2 muestra las etapas del proceso de filtrado de estructuras ontológicas. El usuario ingresa la estructura ontológica a filtrar y obtiene como salida la estructura ontológica filtrada según las opciones especificadas en el filtro (subestructura). La selección del filtro considera el diseño previo de diversos filtros que el usuario deberá elegir para su aplicación automática. Dentro de las opciones de filtrado se consideraron:

(i) Filtrado por función molecular, con el objetivo de obtener una subestructura ontológica que contenga las actividades que ocurren en el nivel molecular.

(ii) Filtrado por componente celular, para obtener aquellos términos que refieren a los elementos de una célula.

(iii) Filtrado por proceso biológico, con el fin de ver la rama de la estructura ontológica que se corresponde con un conjunto de funciones moleculares que forman un proceso biológico.

Debemos notar que en el ámbito de esta tesina nos centramos, sin limitarnos, en el filtro de función molecular puesto que las funcionalidades realizadas por las proteínas son de gran interés para los biólogos que realizan anotaciones de organismos no modelo. En este contexto se intenta contribuir con el proyecto PICT[29] realizado en forma conjunta entre el Cifasis y el INTA de Castelar.

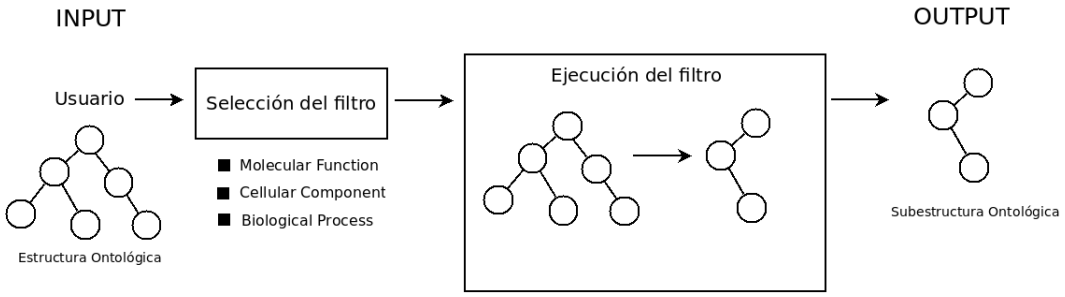

Figura 2: Filtrado de estructuras ontológicas.

[29]PICT PRH nº 253 "Clasificación de Datos Complejos Mediante Integración Borrosa".

El filtro de ontologías es transparente al usuario, de manera que éste no percibe los procesos internos de filtrado. Sobre la especificación del filtro de ontologías volveremos en el Capítulo 5.

4.2. Diseño del Filtro de Bases de Datos

En la exploración de herramientas existentes, observamos que AmiGO y QuickGO permitían el filtrado de un conjunto reducido predefinido de bases de datos y con filtros fijos predefinidos. Por otra parte, con la herramienta GOOSE podíamos armar un filtro a partir de sentencias SQL, pero también se planteaba sobre un conjunto dado de DBs. Asimismo, debemos notar que la generación de consultas SQL no es sencilla para los biólogos, y además, no hay un chequeo de consistencia entre los datos obtenidos después de una consulta y la subestructura ontológica a la cual deberían estar asociados.

Considerando la importancia tanto en la consistencia como en la interpretabilidad del proceso completo en las dos etapas de filtrado: subestructura e instancias de DBs, se estudió la posibilidad de tomar como punto de partida del filtrado de instancias en las DBs, la subestructura ontológica. En base a este estudio se encontró un *script* llamado *map2slim*[30] de *go-perl* que permite mapear las anotaciones[31] a los términos de la ontología filtrada a partir de los siguientes elementos de entrada: una subestructura ontológica, la estructura ontológica que la contiene (ontología completa) y los datos biológicos con sus correspondientes términos GO (anotaciones). Proponemos la adaptación de map2slim como parte del procesamiento de filtrado de las bases de datos. Asimismo, incluiremos opciones de filtrado relevantes para los biólogos llamadas códigos de evidencia: IDA: Inferred from Direct Assay, EXP: Inferred from Experiment, IMP: Inferred from Mutant Phenotype, IEP: Inferred from Expression Pattern, ISS: Inferred from Sequence or Structural Similarity.

De acuerdo a lo expuesto sugerimos los siguientes requerimientos para el filtro de bases de datos:

(i) Diseñar un filtro con opciones de selección de códigos de evidencia en base al *script* map2slim cuyas entradas serán: la estructura ontológica completa, la estructura ontológica filtrada (subestructura) y la DB de anotaciones ingresadas por el usuario.

(ii) Retornar un archivo de salida que contenga el subconjunto de anotaciones, con el código de evidencia seleccionado, que se asocien con los términos de la subestructura.

[30] Para más información consultar: `http://search.cpan.org/~cmungall/go-perl-0.14/scripts/map2slim`, Fecha de consulta: Agosto 2013

[31] Recordamos que las anotaciones vinculan los datos biológicos con términos GO.

La Fig. 3 muestra el diseño del filtro de bases de datos. El proceso de filtrado de instancias asociadas a la subestructura tiene como entrada una subestructura ontológica, la estructura ontológica que la contiene y una DB ingresada por el usuario, y tiene como salida a las instancias de la DB de entrada, asociadas a la subestructura ontológica.

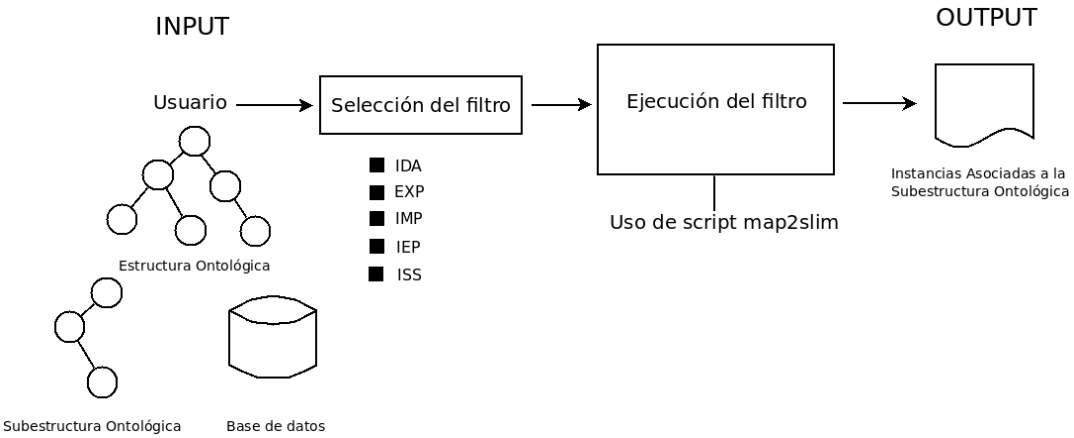

Figura 3: Filtrado de bases de datos.

Análogo al filtro de ontologías, el filtro de bases de datos también es transparente al usuario, de manera que éste no percibe los procesos internos de filtrado. Sobre la especificación del filtro de bases de datos volveremos en el Capítulo 5.

En las dos secciones previas (4.1 y 4.2), hemos definido los diseños de ambos procesos de filtrado por separado pero en vista de una posterior integración. A continuación, mostraremos el diseño del filtrado completo a través de una arquitectura tipo *pipeline* que integra el filtro de ontologías y el filtro de bases de datos en una única tarea.

4.3. Diseño del Pipeline de Filtrado

El proceso de filtrado completo se conformará a partir de la integración del filtro de estructuras ontológicas y del filtro de bases de datos. Para facilitar la tarea del usuario y, considerando que el proceso completo sea transparente, se proporcionará una sola interfaz gráfica donde el usuario pueda elegir la opción de filtrado de la estructura e ingresar la DB de donde quiere extraer las instancias. Con este objetivo se seleccionó la herramienta OBO-Edit como punto de partida, realizando los cambios necesarios, tanto en su interfaz como en el código fuente, para añadir una nueva funcionalidad: el filtrado de bases de datos. De esta manera se logra a partir de una única herramienta las siguientes funcionalidades:

- Creación y edición de estructuras ontológicas,
- Configuración y ejecución de filtros de estructuras ontológicas,
- Configuración y ejecución de filtros de bases de datos a partir de subestructuras ontológicas.

En la Fig. 4 mostramos el proceso de filtrado completo. El diseño del *pipeline* de filtrado se basa en la Arquitectura de Software Tubos y Filtros.

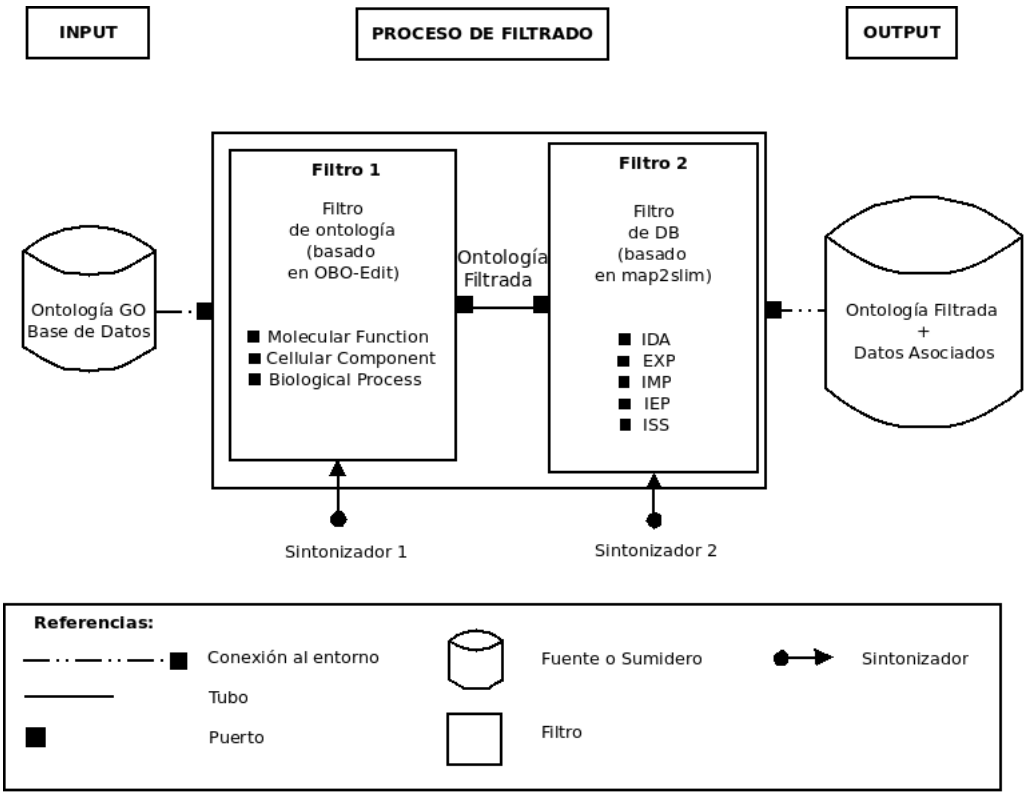

Figura 4: Pipeline de filtrado de estructura ontológica GO y obtención de instancias asociadas a una DB siguiendo la Arquitectura de Software "Tubos y Filtros".

Seguidamente explicamos los distintos elementos del proceso, acompañando cada paso con un pequeño ejemplo gráfico de lo que deseamos obtener como resultado.

4.3.1. Entrada

Las entradas del *pipeline* serán: una ontología en formato OBO, a la que se le podrá aplicar el Filtro 1; y una base de datos con sus términos GO, a

la que se le podrá aplicar el Filtro 2. Por simplicidad, en esta tesina utilizaremos la base de datos de Arabidopsis thaliana. Sin embargo, podría utilizarse cualquier base de datos que siga el siguiente formato general definido por el Consorcio GO y especificado en: `http://www.geneontology.org/GO.format.annotation.shtml`. Las anotaciones se pueden descargar desde: `http://www.geneontology.org/GO.downloads.annotations.shtml` y presentan distintas extensiones, por ejemplo: PAMGO (Plant-Associated Microbe Gene Ontology), TAIR (The Arabidopsis Information Resource), AspGD (Aspergillus Genome Database), entre otras.

En relación a la ontología, podría utilizarse cualquier ontología definida por Gene Ontology (GO). Debemos recordar, que el grupo GO es la mayor iniciativa bioinformática para la estandarización de la representación de conceptos en torno a los genes y productos genéticos en todas las especies. Dentro de GO utilizaremos gene_ontology.1_2.obo[32] en formato OBO versión 1.2. La Fig. 5 muestra su diseño con OBO-Edit donde se puede apreciar, por simplicidad enfocada en cellular component, la vista en modo grafo y en modo árbol.

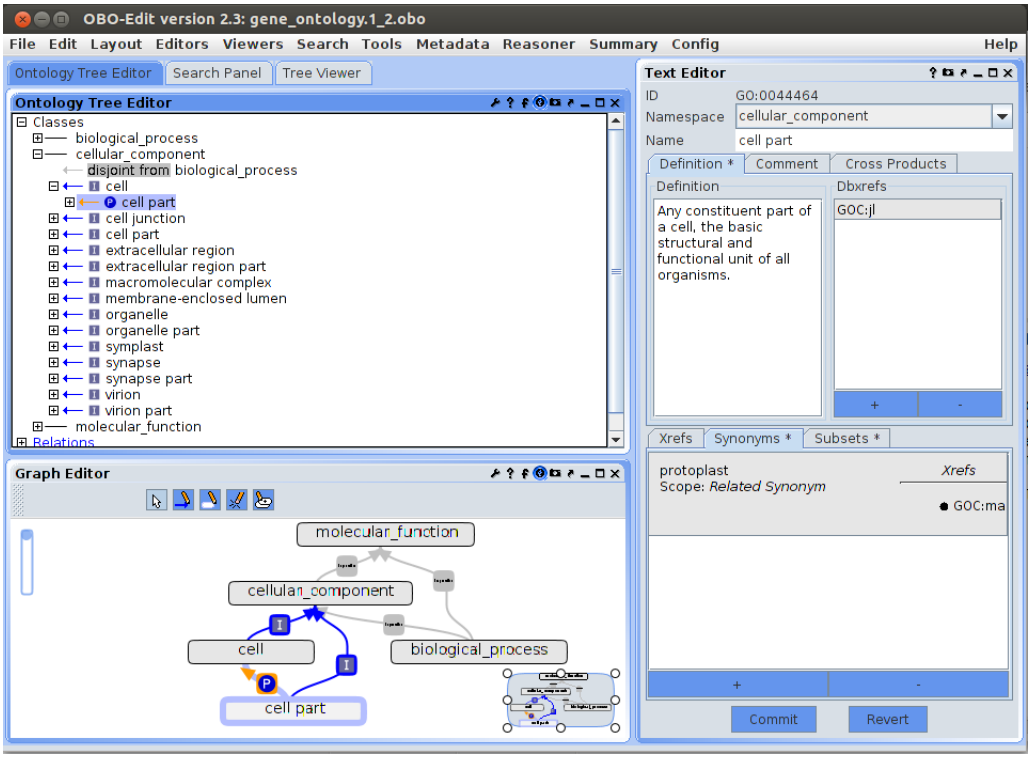

Figura 5: Ontología editada con OBO-Edit.

[32]La ontología gene_ontology.1_2.obo se puede descargar desde: `http://www.geneontology.org/ontology/obo_format_1_2/gene_ontology.1_2.obo`, Fecha de consulta: Agosto 2013

Un ejemplo de base de datos de anotaciones es la denominada *Arabidopsis thaliana*[33] de la que mostraremos en la Fig. 6 una porción a título ilustrativo.

DB	DB Object ID	DB Object Symbol	Qualifier	GO ID	DB:Reference	Evidence	'With' o 'From'	Aspect	DB Object Name	DB Object Synonym	DB Object Type	Taxon	Date	Assigned by
TAIR	locus:2031476	ENO1		GO:0000015	TAIR:Analysis	IEA	INTERPRO:IP	C	enolase 1	AT1G74030|AT1G740	protein	taxon:3702	20120418	TAIR
TAIR	locus:2043067	ENOC		GO:0000015	TAIR:Analysis	IEA	INTERPRO:IP	C	cytosolic enolase	AT2G29560|AT2G295	protein	taxon:3702	20120418	TAIR
TAIR	locus:2044851	LOS2		GO:0000015	TAIR:Analysis	IEA	INTERPRO:IP	C	LOW EXPRESSI	AT2G36530|ENO2|EN	protein	taxon:3702	20120418	TAIR
TAIR	locus:2193997	P40		GO:0000028	TAIR:Commur	IBA	PANTHER:PT	P	AT1G72370	AT1G72370|P40|AP40	protein	taxon:3702	20110729	RefGenome
TAIR	locus:2058324	AT2G04390		GO:0000028	TAIR:Commur	IBA	PANTHER:PT	P	AT2G04390	AT2G04390|T1O3.20|	protein	taxon:3702	20110729	RefGenome
TAIR	locus:2051229	AT2G05220		GO:0000028	TAIR:Commur	IBA	PANTHER:PT	P	AT2G05220	AT2G05220|F5G3.12|	protein	taxon:3702	20110729	RefGenome
TAIR	locus:2084988	RPSAb		GO:0000028	TAIR:Commur	IBA	PANTHER:PT	P	AT3G04770	AT3G04770|RPSAb|4	protein	taxon:3702	20110729	RefGenome
TAIR	locus:2075735	AT3G10610		GO:0000028	TAIR:Commur	IBA	PANTHER:PT	P	AT3G10610	AT3G10610|F13M14.|	protein	taxon:3702	20110729	RefGenome
TAIR	locus:2175428	AT5G04800		GO:0000028	TAIR:Commur	IBA	PANTHER:PT	P	AT5G04800	AT5G04800|MUK11.1|	protein	taxon:3702	20110729	RefGenome
TAIR	locus:2185485	AT5G14850		GO:0000030	TAIR:Commur	ISS	NCBI_gi:1552	F	AT5G14850	AT5G14850|T9L3.150|	protein	taxon:3702	20021003	TIGR
TAIR	locus:5049547	PNT1		GO:0000030	TAIR:Commur	ISS	NCBI_gi:1141	F	AT5G22130	AT5G22130|PNT1|PE	protein	taxon:3702	20021002	TIGR
TAIR	locus:2201786	FATB		GO:0000036	TAIR:Commur	ISS	INTERPRO:IP	F	AT1G08510	AT1G08510|FATB|fatt	protein	taxon:3702	20030905	TIGR
TAIR	locus:2206300	mtACP2		GO:0000036	TAIR:Commur	ISS	INTERPRO:IP	F	AT1G65290	AT1G65290|mtACP2|I	protein	taxon:3702	20030829	TIGR
TAIR	locus:2042331	MTACP-1		GO:0000036	TAIR:Commur	ISS	INTERPRO:IP	F	AT2G44620	AT2G44620|MTACP-1	protein	taxon:3702	20030829	TIGR
TAIR	locus:2090285	FaTA		GO:0000036	TAIR:Commur	ISS	INTERPRO:IP	F	AT3G25110	AT3G25110|AtFaTA|F	protein	taxon:3702	20030905	TIGR
TAIR	locus:2123256	AT4G13050		GO:0000036	TAIR:Commur	ISS	INTERPRO:IP	F	AT4G13050	AT4G13050|F25G13.1	protein	taxon:3702	20030905	TIGR
TAIR	locus:2181216	ACP5		GO:0000036	TAIR:Commur	ISS	INTERPRO:IP	F	AT5G27200	AT5G27200|ACP5|acy	protein	taxon:3702	20030829	TIGR
TAIR	locus:2168968	mtACP3		GO:0000036	TAIR:Commur	ISS	INTERPRO:IP	F	AT5G47630	AT5G47630|mtACP3|I	protein	taxon:3702	20030829	TIGR
TAIR	locus:2156922	ATGRIP		GO:0000042	TAIR:Analysis	IEA	INTERPRO:IP	P		AT5G66030|GRIP|AT5	protein	taxon:3702	20120418	TAIR
TAIR	locus:2156922	ATGRIP		GO:0000042	TAIR:Analysis	IEA	INTERPRO:IP	P		AT5G66030|GRIP|AT5	protein	taxon:3702	20120418	TAIR
TAIR	locus:2020098	ATG12A		GO:0000045	TAIR:Analysis	IEA	INTERPRO:IP	P	AUTOPHAGY 12	AT1G54210|APG12|A	protein	taxon:3702	20120418	TAIR
TAIR	locus:2020098	ATG12A		GO:0000045	TAIR:Analysis	IEA	INTERPRO:IP	P	AUTOPHAGY 12	AT1G54210|AT1G542	protein	taxon:3702	20120418	TAIR
TAIR	locus:2088182	APG12B		GO:0000045	TAIR:Analysis	IEA	INTERPRO:IP	P	AUTOPHAGY 12	AT3G13970|APG12|A	protein	taxon:3702	20120418	TAIR
TAIR	locus:2030280	PSD		GO:0000049	TAIR:Commur	ISS	Pfam:PF04150	F	AT1G72560	AT1G72560|PSD|PAU	protein	taxon:3702	20031003	TIGR
TAIR	locus:2199772	AT1G76170		GO:0000049	TAIR:Analysis	IEA	INTERPRO:IP	F		AT1G76170|AT1G761	protein	taxon:3702	20120418	TAIR
TAIR	locus:2064801	AT2G40660		GO:0000049	TAIR:Analysis	IEA	INTERPRO:IP	F		AT2G40660|AT2G406	protein	taxon:3702	20120418	TAIR
TAIR	locus:2064801	AT2G40660		GO:0000049	TAIR:Commur	ISS	INTERPRO:IP	F	AT2G40660	AT2G40660|T7D17.16	protein	taxon:3702	20030905	TIGR
TAIR	locus:2064821	CTEXP		GO:0000049	TAIR:Publicati	IDA		F	AT2G40730	AT2G40730|CTEXP|c	protein	taxon:3702	20120106	TAIR
TAIR	locus:2050620	ROL5		GO:0000049	TAIR:Analysis	IEA	INTERPRO:IP	F	repressor of lrx1	AT2G44270|AT2G442	protein	taxon:3702	20120418	TAIR

Figura 6: Fragmento de la anotación *Arabidopsis thaliana*.

La base de datos *Arabidopsis thaliana* es un archivo de extensión TAIR (The Arabidopsis Information Resource[34]). Usa el formato estándar del consorcio GO para archivos de asociaciones de genes[35] cuyas columnas son descriptas a continuación[36]:

1: **DB**, base de datos que contribuyó en el armado del archivo (en este caso es siempre "TAIR").

2: **DB_Object_ID**, identificador único para genes de TAIR.

3: **DB_Object_Symbol**, tiene un nombre simbólico de un gen (por ejemplo, "AP2", "AG"). De no existir, estará presente el nombre "AGI" (por ejemplo, "AT1g01010.1").

4: **Qualifier** (opcional), uno o más (separados por pipe '|') de 'NOT', 'contributes_to', 'colocalizes_with' como calificador(es) para una anotación GO, cuando sea necesario.

[33] La base de datos *Arabidopsis thaliana* se puede descargar desde: `http://www.geneontology.org/GO.downloads.annotations.shtml`, Fecha de consulta: Agosto 2013

[34] The Arabidopsis Information Resource se encarga de realizar asociaciones de genes. Para más información consultar: `http://www.arabidopsis.org/`, Fecha de consulta: Agosto 2013

[35] La descripción completa del formato de archivos de asociaciones de genes se encuentra en: `http://www.geneontology.org/GO.format.annotation.shtml`, Fecha de consulta: Agosto 2013

[36] Para más información consultar: `http://www.geneontology.org/gene-associations/readme/tair.README`, Fecha de consulta: Agosto 2013

5: **GO ID**, identificador numérico único para el término GO.

6: **DB:Reference** (|DB:Reference), la referencia asociada con la anotación GO.

7: **Evidence**, el código de evidencia para la anotación GO.

8: **'With' o 'From'** (opcional), cualquier calificador Con o Desde para la anotación GO.

9: **Aspect**, a qué ontología pertenece el término GO (Función, Proceso o Componente).

10: **DB_Object_Name** (|Name) (opcional), un nombre para el producto genético en palabras, por ejemplo, 'acid phosphatase'

11: **DB_Object_Synonym** (|Synonym) (opcional). El "Locus"[37] (por ejemplo, "AT1g01010", "SGR5", "ICX1") es siempre parte de esta información. Para aquellos genes donde sólo el loci genético es conocido, tal como "SGR5", el nombre del gen también está representado como el nombre locus. Cualquier otro nombre (excepto el nombre simbólico, que si existe estará en la columna 3), incluyendo alias usados para el gen, estarán también presentes en esta columna.

12: **DB_Object_Type**, tipo de objeto anotado, por ejemplo, gen, proteína, etc.

13: **Taxon** (|taxon), identificador taxonómico del producto genético que codifica la especie.

14: **Date**, la fecha en que se realizó la anotación GO en el formato.

15: **Assigned_by**, fuente de la anotación ("TAIR", "TIGR")

Dentro de los datos nos enfocamos en la columna número 5 (GO ID) que corresponde a los términos GO. En la figura vemos los términos GO:0000015, GO:0000028, GO:0000030, GO:0000036, GO:0000042, GO:0000045, GO:0000049. Si buscamos dichos términos en: `http://amigo.geneontology.org/`, vemos que corresponden a:

- GO:0000015 phosphopyruvate hydratase complex (Cellular Component)
- GO:0000028 ribosomal small subunit assembly (Biological Process)
- GO:0000030 mannosyltransferase activity (Molecular Function)

[37] En genética, el locus (plural:loci) es el lugar específico del gen o secuencia ADN en un cromosoma.

- GO:0000036 ACP phosphopantetheine attachment site binding involved in fatty acid biosynthetic process (Molecular Function)
- GO:0000042 protein targeting to Golgi (Biological Process)
- GO:0000045 autophagic vacuole assembly (Biological Process)
- GO:0000049 tRNA binding (Molecular Function)

Ilustramos a modo de ejemplo la descripción del término GO:0000015, mostrando la información del término y su *inferred tree view* (vista árbol de inferencia del término, en donde se pueden ver sus términos vecinos).

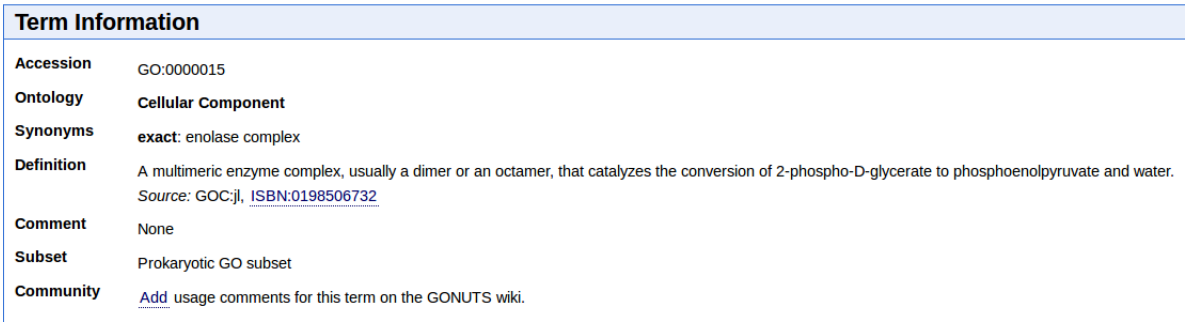

Figura 7: Información del término GO:0000015.

La Fig. 7 muestra información adicional del término formada por su ID, estructura ontológica, sinónimos, definición, entre otros. Podemos apreciar que el término con ID GO:0000015 corresponde a la rama GO denominada componente celular, y se define como una enzima compleja.

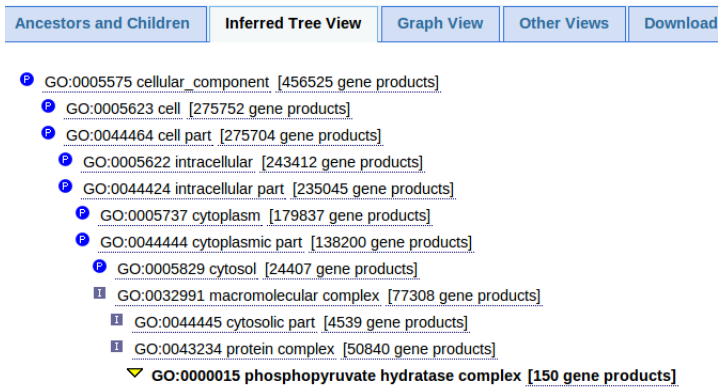

Figura 8: Árbol de inferencia del término GO:0000015.

En la Fig. 8 se ilustra la vista árbol, que muestra de dónde proviene el término, partiendo de la raíz: desde el componente celular hasta la proteína compleja pasando por la célula y el citoplasma, entre otros.

4.3.2. Proceso de Filtrado

Como ya hemos mencionado, el proceso de filtrado se ejecuta en dos etapas. La primera se encarga de filtrar la estructura ontológica (Filtro 1) para obtener la subestructura ontológica filtrada por la opción elegida por el usuario: función molecular, componente celular, proceso biológico (Sección 4.1), donde elegimos la opción de filtrado por la función molecular para la ilustración del método. La segunda etapa se encarga de filtrar la base de datos (Filtro 2) para obtener los datos asociados a la estructura ontológica filtrada en la primera parte. Las opciones en este filtro son: IDA, EXP, IMP, IEP, ISS (Sección 4.2). Estas dos partes se logran a partir de procesos distintos, aunque en cascada: se debe primero filtrar la estructura ontológica para luego filtrar la base de datos y así obtener los datos asociados a la estructura ontológica previamente filtrada.

Siguiendo el ejemplo del apartado anterior, mostramos en la Fig. 9 la subestructura ontológica resultante luego de aplicar el Filtro 1 (con la selección de la opción de filtrado por la función molecular):

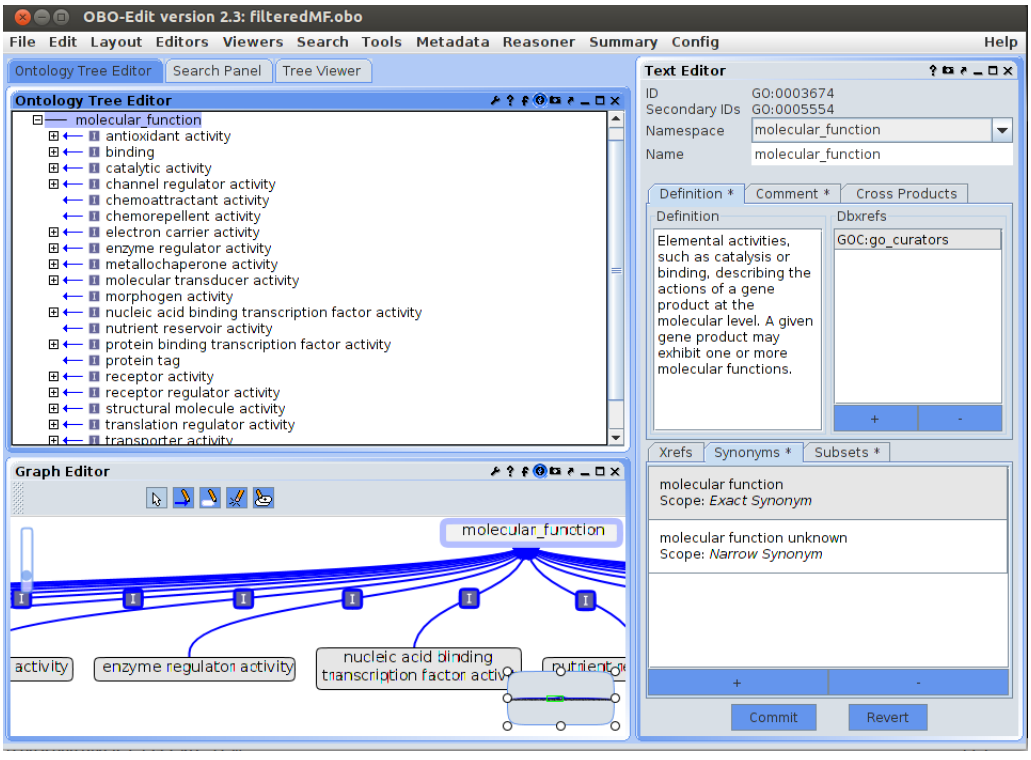

Figura 9: Ontología filtrada por la función molecular editada con OBO-Edit.

Se aplicará luego el Filtro 2 para obtener el subconjunto de instancias de la base de datos *Arabidopsis thaliana* que están asociadas a la subestructura

ontológica previamente obtenida, con el código de evidencia IDA (Inferred from Direct Assay) e ISS (Inferred from Sequence or Structural Similarity). Analizando los términos de la base de datos del ejemplo, podemos ver que los términos GO:0000030, GO:0000036 y GO:0000049 corresponden a molecular function, mientras que el término GO:0000015 corresponde a cellular component y los restantes términos (GO:0000028, GO:0000042 y GO:0000045) a biological process. Por este motivo, luego de aplicar el Filtro 2 debemos obtener los términos GO:0000030, GO:0000036 y GO:0000049. Y de dichos términos, nos quedaremos sólo con aquellos cuyo código de evidencia corresponda a IDA o a ISS.

En la Fig. 10 podemos ver un fragmento de la base de datos *Arabidopsis thaliana* luego de aplicarle el Filtro 2 (mostramos sólo las columnas 1-8). Si nos centramos en la columna 5, notamos que se han eliminado aquellos términos que no correspondían a la función molecular, como eran los términos: GO:0000015 (componente celular), GO:0000028, GO:0000042 y GO:0000045 (proceso biológico). Por el contrario, sí aparecen los términos GO:0000030, GO:0000036 y GO:0000049, correspondientes a la función molecular; y de esos términos sólo aparecen los que tienen código de evidencia IDA o ISS.

TAIR	locus:2185485	AT5G14850		GO:0000030	TAIR:Communication:501714663	ISS
TAIR	locus:504954755	PNT1		GO:0000030	TAIR:Communication:501714663	ISS
TAIR	locus:2201786	FATB		GO:0000036	TAIR:Communication:501714663	ISS
TAIR	locus:2206300	mtACP2		GO:0000036	TAIR:Communication:501714663	ISS
TAIR	locus:2042331	MTACP-1		GO:0000036	TAIR:Communication:501714663	ISS
TAIR	locus:2090285	FaTA		GO:0000036	TAIR:Communication:501714663	ISS
TAIR	locus:2123256	AT4G13050		GO:0000036	TAIR:Communication:501714663	ISS
TAIR	locus:2181216	ACP5		GO:0000036	TAIR:Communication:501714663	ISS
TAIR	locus:2168968	mtACP3		GO:0000036	TAIR:Communication:501714663	ISS
TAIR	locus:2030280	PSD		GO:0000049	TAIR:Communication:501714663	ISS
TAIR	locus:2064801	AT2G40660		GO:0000049	TAIR:Communication:501714663	ISS
TAIR	locus:2064821	CTEXP		GO:0000049	TAIR:Publication:501746929	IDA

Figura 10: Fragmento de la anotación de *Arabidopsis thaliana* luego de aplicarle el Filtro 2 con opciones IDA e ISS.

4.3.3. Salida

Luego del filtrado completo, obtendremos como salida, por un lado, la estructura ontológica filtrada por el Filtro 1, guardada en un archivo en formato `.obo`; y por otro, la base de datos filtrada según el Filtro 2, guardada en un archivo en formato `.csv`. La base de datos final contendrá los datos asociados a la estructura ontológica filtrada previamente.

4.3.4. Sintonizadores

El *pipeline* posee dos sintonizadores[38], por medio de ellos el usuario ingresará los datos necesarios para poder aplicar los filtros. El primer sinto-

[38]Un sintonizador es un procedimiento que permite alterar o sintonizar el comportamiento del filtro. Los sintonizadores sólo pueden ser invocados por componentes que no sean filtros del sistema, generalmente por un componente que implementa una GUI.

nizador permite que el usuario seleccione alguna de las opciones de filtrado que se ofrecen para aplicar a la estructura ontológica editada previamente con OBO-Edit. Dentro de las opciones de filtrado se consideraron: filtrado por función molecular, filtrado por componente celular y filtrado por proceso biológico. El segundo sintonizador permite ingresar una anotación y seleccionar códigos de evidencia de datos. La anotación será filtrada por el Filtro 2 obteniendo los datos asociados a la subestructura ontológica que contengan los códigos de evidencia elegidos.

Gracias a los sintonizadores logramos la flexibilidad en el proceso y la posibilidad de incluir nuevas funcionalidades futuras, permitiendo al usuario poder participar de las etapas, tanto para elegir opciones como para ingresar los datos requeridos en el momento que sea necesario.

4.4. Discusión

En lo que respecta a este capítulo, hemos desarrollado los diseños para cada uno de los filtros por separado: filtro de ontologías y filtro de bases de datos. A partir del análisis y exploración de herramientas realizado, decidimos utilizar la herramienta OBO-Edit para incorporar estas dos nuevas funcionalidades abocadas a procesos de filtrado, por ser esta herramienta la más completa en el ámbito de la bioinformática para crear, editar y filtrar estructuras ontológicas con filtros configurados manualmente. Las dos nuevas funcionalidades agregadas son: filtrado automático de estructuras ontológicas y filtrado de bases de datos. Ambos filtros flexibles y transparentes al usuario. Además de los diseños de los filtros de forma independiente, finalizamos el capítulo con el diseño del *pipeline* de filtrado basado en la Arquitectura de Tubos y Filtros, mediante el cual se logra la unificación de los dos procesos de filtrado en un único proceso flexible y transparente. Incorporamos también el *pipeline* de filtrado a OBO-Edit.

A continuación detallamos la especificación de los procesos de filtrado, mostrando los casos de uso para cada etapa.

5. Especificación

El objetivo principal de este capítulo es especificar los módulos definidos para el filtrado de estructuras ontológicas, filtrado de bases de datos y *pipeline* de filtrado, teniendo en cuenta los requerimientos y los diseños presentados en el Capítulo 4. La especificación tendrá una descripción de los resultados a obtener desde un punto de vista funcional.

5.1. Caracterización de Actores

Ilustramos la caracterización de los actores con el siguiente diagrama:

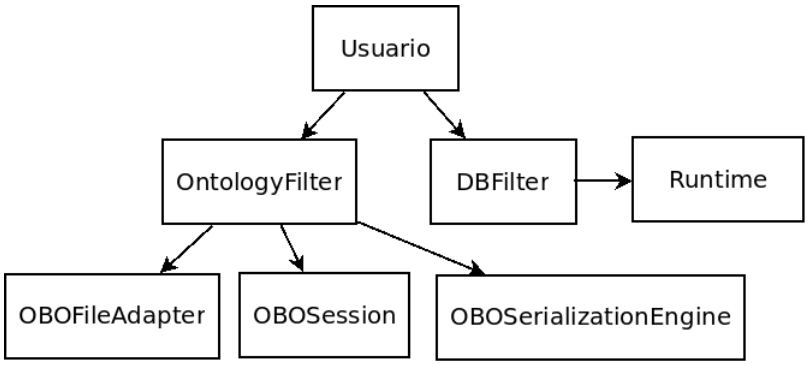

Figura 11: Actores y relaciones entre ellos.

En la figura podemos apreciar los diferentes actores y cómo interactúan entre sí. A continuación haremos una breve descripción de los roles de cada uno.

1- Usuario

El usuario ingresa una estructura ontológica y selecciona una opción de filtrado entre las proporcionadas por OBO-Edit. Ingresa también una base de datos asociada a la estructura ontológica y selecciona códigos de evidencia.

Resultado esperado: instancias de la base de datos asociadas a la estructura ontológica filtrada según opciones elegidas por el Usuario.

2- Módulo OntologyFilter

Ejecuta el filtro de la estructura ontológica según la opción elegida por el Usuario a través de los módulos `OBOFileAdapter`, `OBOSession` y `OBOSerilizationEngine`. Resultado informado al Usuario: grafo representado a través de GO_TERMs (clases GO) y relaciones en formato `.obo`.

3- Módulo DBFilter

Ejecuta el filtrado de bases de datos según hayan sido los datos ingresados por el Usuario.

Resultado informado al Usuario: registros de datos asociados a los GO_TERMs del OntologyFilter en la DB ingresada en formato `.csv`.

4- Módulo OBOFileAdapter

Adaptador para manejo de archivos en formato `.obo`. Contiene la información del archivo ingresado por el Usuario (que contiene la estructura ontológica) como por ejemplo: metadata, nombre, formato, ruta. Permite realizar operaciones sobre el archivo, tanto de escritura como de lectura.

5- Módulo OBOSession

Contiene los datos de la sesión iniciada en OBO-Edit: usuario, espacio de nombres, historial, entre otros. La operación de lectura o escritura llevada a cabo por el OBOFileAdapter retorna una OBOSession.

6- Módulo OBOSerializationEngine

Motor para escribir archivos OBO. Provee funciones para configurar y ejecutar un filtro de ontologías. Se utiliza para ejecutar el filtro de ontologías con las configuraciones correspondientes a la opción elegida por el Usuario.

7- Módulo Runtime

Ejecuta *scripts* creados para el filtrado de bases de datos, utilizando los datos de entrada necesarios. Se utiliza para ejecutar el *script* map2slim con el objetivo de filtrar la base de datos ingresada por el Usuario para obtener aquellas instancias asociadas a la estructura ontológica previamente filtrada.

5.2. Casos de Uso

Describimos los diferentes casos de uso indicando: actor que interviene, precondición, próposito, casos de éxito, casos de falla y escenario de éxito.

1- Filtro de Ontologías

Actor	Usuario
Precondición	Se ha cargado un archivo que contiene la ontologia a filtrar.
Propósito	Filtrar una ontología ingresada.
Casos de Éxito	El Usuario obtiene un archivo en formato `.obo` que contiene la ontología filtrada.
Casos de Falla	Se produce un fallo en el acceso al archivo fuente. Se produce un fallo en la escritura del archivo.
Escenario de Éxito	Paso 1: El Usuario elige y aplica el filtro automático. Paso 2: OntologyFilter invoca a los módulos OBOFileAdapter, OBOSession y OBOSerializationEngine para filtrar la ontología. Luego, informa al Usuario que se realizó el filtro exitosamente indicando el nombre del archivo `.obo` resultante.

2- Filtro de Bases de Datos

Actor	Usuario
Precondición	Se ha instalado el paquete go-perl. Se ha filtrado una estructura ontológica. Se ha ingresado una base de datos.
Propósito	Filtrar una base de datos ingresada.
Casos de Éxito	El Usuario obtiene un archivo en formato `.csv` que contiene la base de datos filtrada.
Casos de Falla	Se produce un fallo en el acceso a la base de datos. Se produce un fallo en la escritura del archivo.
Escenario de Éxito	Paso 1: El Usuario ingresa la ontología completa, la ontología filtrada, la base de datos, selecciona los códigos de evidencia y aplica el filtro automático. Paso 2: DBFilter invoca al módulo Runtime para filtrar la base de datos. Luego, filtra el resultado según los códigos de evidencia elegidos e informa al Usuario que se realizó el filtro exitosamente indicando el nombre del archivo `.csv` resultante.

3- Pipeline de Filtrado

Actor	Usuario
Precondición	Se ha cargado un archivo que contiene la ontología a filtrar. Se ha instalado el paquete go-perl. Se ha ingresado una base de datos.
Propósito	Filtrar una ontología ingresada. Filtrar una base de datos ingresada.
Casos de Éxito	El Usuario obtiene un archivo en formato `.obo` que contiene la ontología filtrada. El Usuario obtiene un archivo en formato `.csv` que contiene la base de datos filtrada.
Casos de Falla	Se produce un fallo en los accesos a la ontología o a la base de datos. Se produce un fallo en la escritura de alguno o ambos archivos.
Escenario de Éxito	Paso 1: El Usuario elige el filtro de ontologías, ingresa la base de datos, selecciona los códigos de evidencia y aplica el filtro automático. Paso 2: OntologyFilter invoca a los módulos OBOFileAdapter, OBOSession y OBOSerializationEngine para filtrar la ontología. Luego, informa al Usuario que se realizó el filtro exitosamente indicando el nombre del archivo `.obo` resultante. Paso 3: DBFilter invoca al módulo Runtime para filtrar la base de datos. Luego, filtra el resultado según los códigos de evidencia elegidos e informa al Usuario que se realizó el filtro exitosamente indicando el nombre del archivo `.csv` resultante.

5.3. Discusión

En el siguiente capítulo nos abocamos a la implementación de los filtros por separado y del *pipeline* de filtrado, cuyos diseños y especificaciones se mostraron a lo largo de los Capítulos 4 y 5. Mostramos diferentes cuestiones de interés para aquellos que deseen extender el alcance de este trabajo. Detallamos las clases y métodos existentes en los códigos fuente de la herramienta OBO-Edit que fueron modificados y/o agregados al proyecto. Explicamos también el diseño de implementación elegido y aplicado en cada paso, basándonos en la programación orientada a objetos.

6. Aspectos de Implementación

En este capítulo describiremos el proceso de implementación de los filtros por separado y su integración dentro del *pipeline* de filtrado. Primero nos centraremos en el desarrollo de los Filtros 1 y 2 (filtro de ontologías y de bases de datos, respectivamente) y posteriormente en la realización de las GUIs que invocarán a los sintonizadores. Recordamos que el modelo posee dos sintonizadores: uno para que el usuario ingrese la opción de filtrado de estructuras ontológicas, y otro para que el usuario ingrese la base de datos y seleccione código(s) de evidencia. Por último, integraremos los módulos dando lugar al *pipeline* de filtrado: una única GUI que incluye los dos procesos de filtrado. Para más detalle de las clases modificadas y agregadas consultar el Apéndice E (8.5).

Debemos notar que la implementación se realizó considerando las opciones de filtrado manual que poseía OBO-Edit[39], luego de una exploración de las herramientas libres existentes (Capítulo 2). OBO-Edit brinda la clase `OBOSerializationEngine` cuya función `serialize` logra la ejecución de un filtro ontológico según la configuración confeccionada. El filtro se aplica al archivo `.obo` que contiene la ontología GO, el cual fue ingresado por el usuario, y cuyos datos quedan guardados en las configuraciones de un adaptador de tipo `OBOFileAdapter`. Reutilizando estas clases, pudimos armar filtros ontológicos predefinidos para ejecutar de forma automática. Por otro lado, la herramienta no brindaba el filtrado de bases de datos, cuya implementación se realizó utilizando la clase `Runtime` de `Java`[40] para poder ejecutar *scripts*, en nuestro caso ejecutaremos el *script* `map2slim` que describimos en las secciones previas. A continuación detallamos las diferentes etapas llevadas a cabo durante la implementación.

6.1. Filtro 1: Filtro de Ontologías

Diseñamos el Filtro 1 según el paradigma de Programación Orientada a Objetos (POO)[41], de manera tal que exista una interfaz abstracta y cada tipo de Filtro 1 hereda de aquella. De esta manera, dejamos abierta la posibilidad de extender nuestra implementación a distintos tipos de filtro sobre ontologías que se quieran desarrollar. Nuestra implementación abarca solamente el filtrado por la función molecular. Ilustramos el diseño con la siguiente figura:

[39] Para la descarga del programa Eclipse y los códigos fuente de OBO-Edit consultar los Apéndices B y C (8.2 y 8.3), respectivamente.

[40] Para más detalles del lenguaje de programación Java consultar el Apéndice A (8.1).

[41] La Programación Orientada a Objetos (POO) es un paradigma de programación que representa conceptos como objetos, los cuales poseen atributos y métodos.

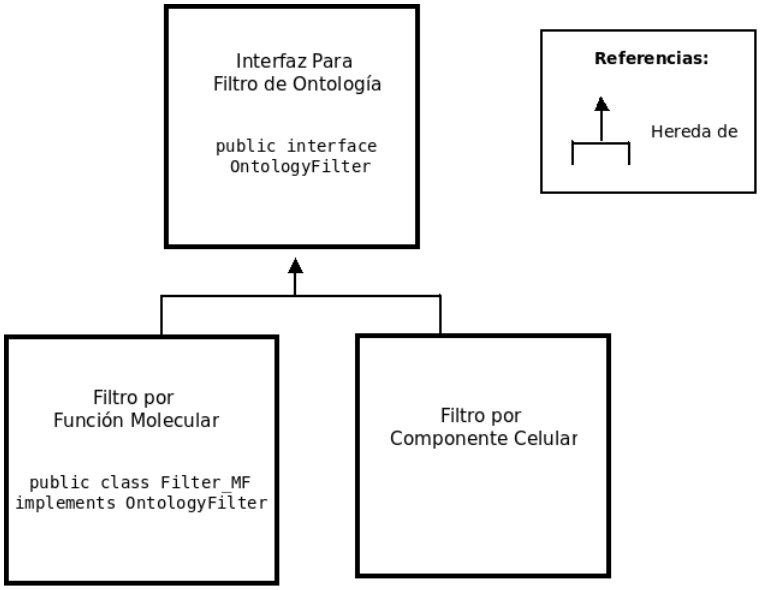

Figura 12: Diseño del Filtro de Ontologías.

Declaramos la interfaz abstracta `OntologyFilter` que se comprondrá de los siguientes métodos:

- `void filterOntology();`
- `Filter<Link> getLinkFilter();`
- `Filter<IdentifiedObject> getTermFilter();`
- `TagFilter getTagFilter();`
- `HashSet<OBOConstants.TagMapping> getTagsToWrite();`
- `String getOntologyPath();`
- `boolean isFiltered();`

La interfaz logra abstraer el proceso de filtrado de la ontología, llevado a cabo por el método `filterOntology`. Los métodos `get` se utilizan para poder obtener los filtros de tipo Link, Term y Tag utilizados en el proceso. El método `getOntologyPath` retorna la ruta al archivo que contiene la ontología GO. El último método de la interfaz sirve para ver el estado del filtro, si pudo o no ser realizado.

Los distintos tipos de filtro de ontologías que se desarrollen deben heredar de esta interfaz. En nuestro caso, la clase `Filter_MF` es quien implementa la interfaz y lleva a cabo el filtro MF. Se pueden añadir todos los filtros que se deseen, como por ejemplo, filtro por componente celular.

6.1.1. Filtro MF

Para la implementación del filtro MF, creamos la clase `Filter_MF`, la cual implementa la interfaz abstracta `OntologyFilter`.

Como primera medida debemos configurar tres tipos de filtro: Tag, Link y Term. Para ello utilizamos las clases previstas por OBO-Edit. A continuación, describimos el armado de los tres filtros (en el Apéndice D (8.4) podemos ver la configuración manual de estos filtros). Posteriormente, mostramos los pasos a seguir para llevar a cabo el filtrado de la ontología a partir de estos tres filtros.

Tag Filter

Para el filtro MF debemos setear todos los tags excepto dos: `consider` y `replaced by`. Para ello, utilizamos la clase `TagFilter` la cual nos provee de las funciones `setReplaced_byTagToBeWritten` y `setConsiderTagToBeWritten` que toman como entrada un valor booleano, en nuestro caso queremos que estén en False. Luego obtenemos los valores de los tags con `getTagsToWrite`.

Link Filter

Armamos el filtro Link utilizando la clase `LinkFilterEditor`. Allí definimos la función `layoutGUI_LinkFilterMF` que se encarga de la configuración del filtro, seteando las variables de la clase para formar: *Find links where Type don't have a Name that contains the value part_of.* Desglosamos la frase y armamos las siguientes variables:

- `aspectBox` seteada en 1 que corresponde a *Type*
- `notBox` seteada en 1 que corresponde a *don't have*
- `criterionBox` seteada en 2 que corresponde a *Name*
- `comparisonBox` seteada en 0 que corresponde a *contains*
- `valueField` seteada en "part_of"que corresponde a *part_of*

Una vez realizado esto, utilizamos la clase `LinkFilterEditorFactory` que nos brinda el método `getFilter` que nos devuelve el filtro que buscamos en una variable de tipo `Filter<Link>`.

Term Filter

De forma análoga armamos el filtro Term (o también llamado Object) utilizando la clase `TermFilterEditor`. Allí definimos la función `layoutGUI_TermFilterMF` que se encarga de la configuración del filtro, seteando las variables de la clase para formar: *Find terms that have a Namespace that equals the value molecular_function.* Desglosamos la frase y armamos las siguientes variables:

- `notBox` seteada en 0 que corresponde a *have*
- `criterionBox` seteada en 8 que corresponde a *Namespace*
- `comparisonBox` seteada en 1 que corresponde a *equals*
- `valueField` seteada en "molecular_function"que corresponde a *molecular_function*

Una vez realizado esto, utilizamos la clase `TermFilterEditorFactory` que nos brinda el método `getFilter` que nos devuelve el filtro que buscamos en una variable de tipo `Filter<IdentifiedObject>`.

Obtenidos los 3 filtros necesarios seguimos los siguientes pasos:

(i) Armamos el adaptador de archivo de formato OBO correspondiente a la clase `OBOFileAdapter`

(ii) Configuramos el adaptador

(iii) Armamos la sesión OBO con la clase `OBOSession`, seteando la operación `WRITE_ONTOLOGY` ya que vamos a escribir un archivo (el que contendrá la ontología filtrada resultante)

(iv) Instanciamos la clase `OBOSerializationEngine`, la cual posee el método `serialize` que es quien se encarga de realizar el filtro según los parámetros configurados: filtros Tag, Term y Link creados previamente, y una dirección de archivo donde se guarda la ontología filtrada. La ruta que utilizamos es: *"../../filteredMF.obo"*

6.2. Filtro 2: Filtro de Bases de Datos

Diseñamos el Filtro 2 de forma análoga al Filtro 1, pudiendo también extender la implementación agregando distintos tipos de filtro de bases de datos. Nuestra implementación de base de datos se basa en el *script* map2slim de go-perl. Además utilizamos la librería *Apache Commons CSV parser* para parsear archivos en formato `.csv` y poder filtrar la anotación por los códigos de evidencia seleccionados. Mostramos el diseño aplicado al filtro de bases de datos a través de la siguiente figura:

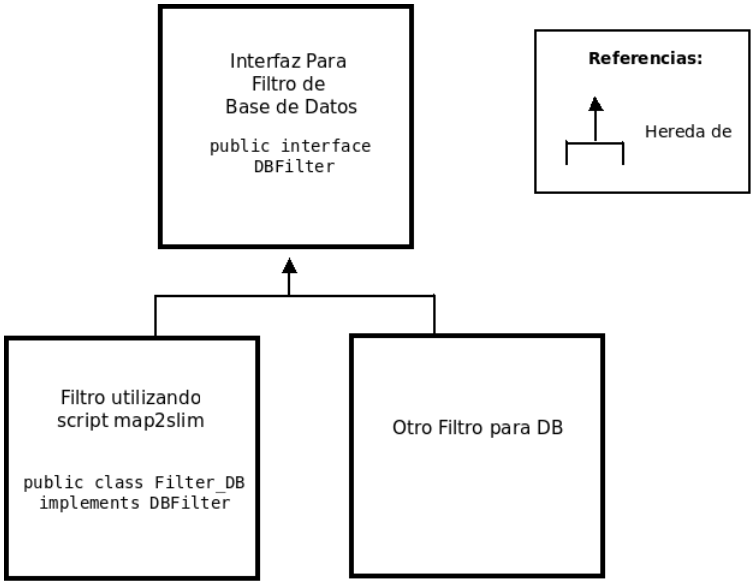

Figura 13: Diseño del Filtro de Bases de Datos.

Declaramos la interfaz abstracta `DBFilter` que se compone de los siguientes métodos:

- `void filterDatabase();`
- `File getOntologyFile();`
- `File getFilterOntologyFile();`
- `File getAnnotationFile();`
- `void browse_ontology();`
- `void browse_filter_ontology();`
- `void browse_annotation();`
- `void setOntologyFile(String path);`
- `void setFilterOntologyFile(String path);`
- `void setEvidenceCode(ArrayList<String> ec_list);`
- `boolean isFiltered();`

La interfaz logra abstraer el proceso de filtrado de la base de datos, llevado a cabo por el método `filterDatabase`. Los distintos tipos de filtro de bases de datos que se desarrollen deben heredar de esta interfaz. Se pueden

añadir más filtros de ser necesario. Para poder acceder a los archivos ingresados por el usuario existen los tres métodos `get` de la interfaz y los archivos se obtienen a través de tres métodos `browse`. Existen dos métodos para poder setear las rutas de los archivos que contienen la estructura ontológica y la estructura ontológica filtrada: `setOntologyFile` y `setFilterOntologyFile`, respectivamente. El método `setEvidenceCode` se utiliza para armar la lista con los códigos de evidencia seleccionados. Al igual que la interfaz anterior, el último método sirve para ver el estado del filtro, si pudo o no ser realizado.

En nuestro caso, la clase `Filter_DB` es la encargada de implementar la interfaz. Para poder realizar este filtro, contamos con el *script* `map2slim` que toma como parámetros de entrada: la ontología completa, la ontología filtrada y la anotación de donde extraer los datos, dando como resultado un archivo que contiene todos los datos de la anotación asociados con la ontología filtrada. Para ello, se necesita que el usuario ingrese los tres archivos correspondientes: ontología filtrada, ontología completa y anotación.

6.2.1. Filtro DB

Para la implementación del filtro DB, creamos la clase `Filter_DB`, la cual implementa la interfaz abstracta `DBFilter`.

El filtro DB que implementamos se vale de tres archivos que ingresa el usuario y del *script* map2slim anteriormente nombrado. Para el seteo de los archivos contamos con las tres funciones **browse**, una para cada archivo necesario: ontología filtrada, ontología completa y anotación. La función que ejecuta el filtro se vale de la clase `Runtime`[42] de Java que permite correr *scripts* gracias a la función `exec`. Luego de obtener este filtro, se parsea el resultado a fines de obtener los datos que contengan los códigos de evidencia elegidos. De no seleccionarse ningún código de evidencia, el archivo que se devuelve es el obtenido al aplicar el *script*. La ruta del archivo donde se guardan los datos asociados a la ontología filtrada es *"../../filteredDB.csv"*.

6.3. Confección de GUIs

Desarrollamos tres interfaces gráficas (GUIs[43]) para interactuar con el usuario. A las mismas se puede acceder desde la barra de menú de OBO-Edit y se denominan: `Ontology Filter`, `Database Filter` y `Pipeline Filter`, para poder filtrar la ontología, la base de datos y realizar un *pipeline* de filtrado, respectivamente. Para agregarlas al menú se modificó la clase `FileMenu`

[42] Para más información de la clase Runtime consultar: `http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Runtime.html`, Fecha de consulta: Septiembre 2013

[43] La interfaz gráfica de usuario, conocida también como GUI (del inglés Graphical User Interface) es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computadora.

de OBO-Edit. A continuación, describimos estas dos pantallas.

6.3.1. Interfaz Gráfica 1: Filtro de Ontologías

Desarrollamos esta interfaz gráfica en la clase `OntologyFilterGUI`, y hace uso de la clase `Filter_MF` instanciándola de la siguiente manera:

```
OntologyFilter mf_filter = new Filter_MF();
```

La siguiente figura muestra la interfaz gráfica para el filtro de ontologías:

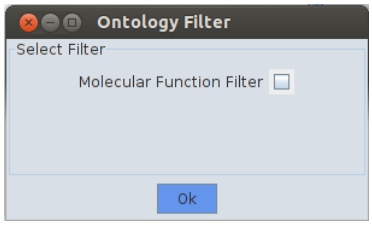

Figura 14: Interfaz Gráfica para el Filtro de Ontologías.

En la pantalla que muestra la figura, podemos seleccionar el filtro a aplicar, en nuestro caso existe un sólo filtro: filtrado por función molecular. Sin embargo, la interfaz es extendible a más filtros como hemos detallado anteriormente. Al presionar el botón `Ok` se ejecuta el filtro y se devuelve el archivo `filteredMF.obo`. De no haber cargado previamente una ontología, el archivo queda sin términos, sólo con el encabezado. Al finalizar el filtrado, aparece un cuadro de diálogo que indica si el filtro fue o no exitoso.

6.3.2. Interfaz Gráfica 2: Filtro de Bases de Datos

Desarrollamos esta interfaz gráfica en la clase `DBFilterGUI`, y hace uso de la clase `Filter_DB` instanciándola de la siguiente manera:

```
DBFilter db_filter = new Filter_DB();
```

La siguiente figura muestra la interfaz gráfica para el filtro de bases de datos:

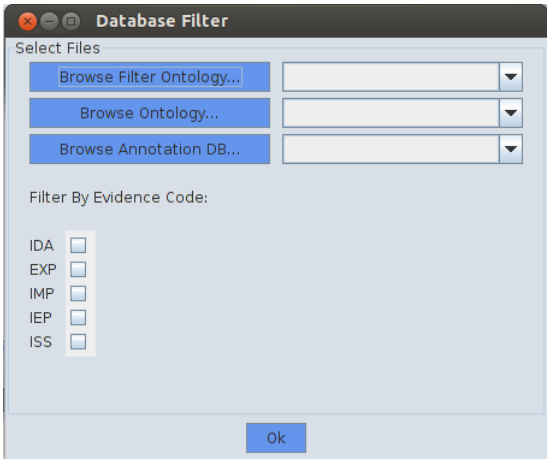

Figura 15: Interfaz Gráfica para el Filtro de Bases de Datos.

El usuario debe ingresar los tres archivos requeridos: la ontología filtrada y la completa, ambas en formato `.obo` y la anotación, que es la que contiene los datos a filtrar. También puede elegir, si desea, uno o más códigos de evidencia por los cuales filtrar la anotación. Al presionar el botón `Ok` se ejecuta el filtro y se devuelve el archivo `filteredDB.csv`. De no poseer instalada la librería `GO::Perl`[44] aparece un cartel de aviso. Al finalizar el filtrado, un cuadro de diálogo indica si el filtro fue o no exitoso.

6.3.3. Interfaz Gráfica 3: Pipeline

Para esta interfaz creamos la clase `PipelineGUI` que se vale de las dos interfaces anteriores instanciándolas:

```
OntologyFilterGUI of_gui = new OntologyFilterGUI();
DBFilterGUI db_gui = new DBFilterGUI();
```

La siguiente figura muestra la interfaz gráfica para el *pipeline* de filtrado:

[44] Para descargar la librería consultar: `http://search.cpan.org/~cmungall/go-perl/`, Fecha de consulta: Agosto 2013

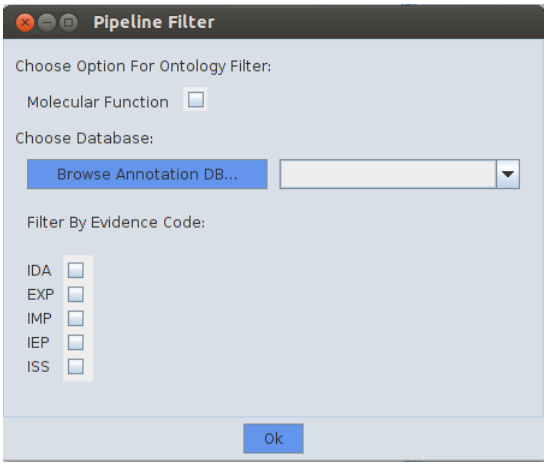

Figura 16: Interfaz Gráfica para el Pipeline de Filtrado.

A partir de esta interfaz se puede lograr un *pipeline* de filtrado, filtrando primero la ontología y luego la base de datos. Como resultado final obtenemos, por un lado, el archivo `filteredMF.obo` que posee la ontología filtrada por la función molecular, y por otro, el archivo `filteredDB.csv` que posee los datos asociados con la ontología filtrada.

El usuario ingresa la opción de filtrado por la que quiere filtrar la estructura ontológica editada previamente. Luego elige la base de datos junto con los códigos de evidencia, de la que quiere recuperar aquellas instancias asociadas que se corresponden con la estructura ontológica filtrada. Al presionar el botón `Ok`, se ejecuta primero el filtro de ontologías y, luego, con el archivo `.obo` obtenido y la base de datos ingresada, se ejecuta el filtro de base de datos. Los cuadros de diálogo que indican la ejecución exitosa o no exitosa de los filtros son los mismos que se utilizan en las interfaces explicadas con anterioridad.

6.4. Discusión

Quedan detallados brevemente en este capítulo algunos aspectos de la implementación de los filtros diseñados y especificados: filtro de ontologías, filtro de bases de datos y *pipeline* de filtrado. Para más detalle de las clases y métodos expuestos anteriormente, podemos consultar los códigos fuente modificados y/o agregados que se encuentran en el Apéndice E (8.5).

7. Conclusión y Trabajos Futuros

En el ámbito de esta tesina se abordó el problema bioinformático de cómo capturar en forma automática y transparente, parte de la estructura ontológica de Gene Ontology junto con instancias asociadas a dicha subestructura proveniente de diversas bases de datos. Este problema estaba motivado por los procesos de predicción de anotaciones electrónicas basados en aprendizaje automatizado supervisado. Específicamente, en contribuir con el diseño de los conjuntos de datos necesarios para la generación de las bases de conocimiento.

Como objetivo se propuso la realización de una herramienta computacional para gestionar el filtrado sobre la estructura ontológica GO, obteniendo también los datos asociados a la subestructura en las bases de datos de anotaciones. El objetivo fue puesto en marcha a partir de un estudio detallado de herramientas existentes para el filtrado de ontologías GO y de bases de datos (Capítulo 2); comparándolas y contrastándolas, extrayendo ventajas y desventajas, centrándonos en los procesos de filtrado que brinda cada una de ellas.

Luego de trazar los objetivos específicos (Capítulo 3), se comenzó con el diseño del filtrado propiamente dicho (Capítulo 4), donde se planteó el proceso de filtrado en dos etapas: *i*) filtrado de estructura ontológica, *ii*) filtrado de base de datos asociada. El primer filtro se basó en OBO-Edit por ser la herramienta más completa para la creación y edición de ontologías en el ámbito de Gene Ontology, donde en esta primera versión se diseñaron las opciones de filtrado por criterio biológico siguiendo las tres ramas de GO: función molecular, componente celular y proceso biológico. Debemos notar que este criterio puede extenderse a más opciones en trabajos futuros. El segundo filtro se basó en el *script* map2slim donde fue necesario además incluir según el criterio biológico, las opciones de selección por códigos de evidencia de la anotación: IDA, EXP, IMP, IEP, ISS[45]. Los dos filtros resultan transparentes, dado que no se perciben procesos internos. Por último, se integraron ambos filtros dentro de un *pipeline* de filtrado, basado en la Arquitectura Tubos y Filtros.

Seguidamente (Capítulo 5), especificamos los filtros diseñados, mostrando los casos de uso posibles, con los actores que intervienen en ellos. Los casos de uso describen principalmente casos de éxito y casos de fallas, útiles para la posterior implementación y/o extensión de los filtros.

Por último (Capítulo 6), describimos algunos aspectos interesantes de la implementación de ambos procesos de filtrado, tanto de manera independiente, como su posterior unificación en un *pipeline* de filtrado integrado a OBO-Edit. De esta manera, la herramienta es capaz de brindar: *i*) Un

[45]Para ver la descripción de los códigos de evidencia consultar: `http://www.geneontology.org/GO.evidence.shtml`, Fecha de consulta: Agosto 2013

proceso de filtrado automático de estructuras ontológicas a partir de una ontología GO ingresada y un tipo de filtro elegido por el usuario, obteniendo la subestructura ontológica correspondiente. *ii*) Un proceso de filtrado automático de bases de datos a partir de una subestructura ontológica, un conjunto de anotaciones, y las opciones de filtrado por códigos de evidencia elegidas por el usuario. El resultado obtenido es el subconjunto de instancias asociadas a dicha subestructura. *iii*) Un *pipeline* de filtrado que logra los procesos de filtrado *i*) y *ii*) en una única tarea.

Contamos entonces con una herramienta computacional que gestiona el filtrado sistemático, flexible y transparente sobre la ontología GO y las bases de datos de anotaciones, que fue el objetivo general del presente trabajo.

La herramienta ha sido mostrada a biólogos quienes la están utilizando actualmente con datos de Helianthus annuus (girasol) y Solanum lycopersicum (tomate). Asimismo se planea proponer la herramienta al grupo de desarrolladores de OBO-Edit. Dentro de este último punto, ya hemos iniciado intercambios vía web a través del siguiente foro `http://sourceforge.net/mailarchive/forum.php?forum_name=geneontology-oboedit-working-group`

7.1. Trabajos Futuros

Consideramos que nuestra tesina está abierta a nuevos aportes. Nuestro trabajo contribuye a facilitar la tarea de los biólogos al añadir a la herramienta OBO-Edit el proceso de filtrado para la generación de conjuntos de datos.

Otro aporte interesante podría ser la automatización de otros filtros ontológicos de uso masivo, además del filtro por función molecular que fue implementado. Por ejemplo, podría realizarse la automatización del filtro por componente celular o proceso biológico. Asimismo, podrían plantearse opciones a filtrar por nivel de profundidad del grafo, dando la posibilidad de trabajar con conceptos más o menos generales. En cuanto a los filtros por nivel, podría implementarse un filtro ontológico iterable, es decir, aplicar distintos tipos de filtro iterando sobre los niveles ontológicos.

En relación al filtrado en las bases de datos, se podrían incorporar a las interfaces gráficas las opciones de filtrado que ofrece map2slim, o nuevos códigos de evidencia para filtrar las anotaciones que sean útiles en el ámbito biológico. También se podría agregar un nuevo tipo de filtrado de bases de datos que permita un filtrado de múltiples bases.

Otro punto que resultaría de gran utilidad, sería el desarrollo de nuevas funcionalidades para OBO-Edit para acceder a las ontologías y/o bases de datos a través de la web. Esto brindaría además de la comodidad de no guardar las ontologías o bases de datos a utilizar de manera local, la posibilidad de trabajar con las últimas versiones de la web. Por último, otro aspecto interesante, sería incorporar a OBO-Edit un editor o visualizador de

las anotaciones, debido a que la herramienta sólo se aboca a las estructuras ontológicas.

Nuestro trabajo se aboca a ontologías y bases de datos dentro del dominio de GO utilizando el editor OBO-Edit, sin embargo, podría extenderse la implementación a otro dominio ontológico a través de las herramientas correspondientes.

8. Apéndices

8.1. Apéndice A: Tecnologías Utilizadas

Tecnología Java y OBO-Edit

En esta sección describimos brevemente la tecnología Java, la cual es la elegida para el desarrollo de la herramienta OBO-Edit.

El lenguaje de programación Java es un lenguaje de alto nivel caracterizado por ser un lenguaje simple, abierto, multiplataforma, orientado a objetos, distribuido, multi-hilo, dinámico, portable, seguro y robusto. Los códigos fuente se escriben en archivos de extensión *.java*. Los archivos son compilados en un archivo *.class* por el compilador javac. Dicho archivo no contiene código nativo al procesador, sino que contiene *bytecodes* que es el lenguaje de la Máquina Virtual Java (JVM). Luego se corre la aplicación con una instancia de la JVM (Fig. 17).

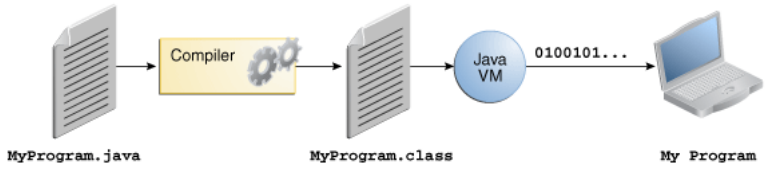

Figura 17: Un diagrama sobre el proceso de desarrollo de software.

La plataforma Java posee dos componentes: la Máquina Virtual Java y la Interfaz de Programación de Aplicaciones (API). La API es un conjunto de componentes de software ya armados que nos proveen de herramientas. Los componentes de software están agrupados en librerías (paquetes) de clases e interfaces relacionadas.

La siguiente figura muestra en conjunto un código fuente, la API, la JVM y la plataforma de hardware:

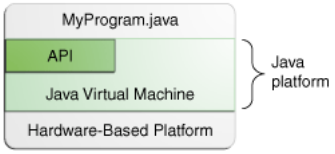

Figura 18: La API y la JVM aíslan al programa del hardware subyacente.

Como existe aquí una plataforma independiente del entorno, la plataforma Java puede ser un poco más lenta que el código nativo. Sin embargo, los avances del compilador y las tecnologías de las máquinas virtuales traen

una performance cercana al código nativo logrando la portabilidad de los sistemas[46].

El proyecto OBO-Edit posee dos APIs: BBOP[47] y OBO.

[46]Para ampliar conocimientos de la Tecnología Java: `http://docs.oracle.com/javase/tutorial/`, Fecha de consulta: Marzo 2013

[47]Página principal de BBOP: `http://berkeleybop.org/`, Fecha de consulta: Marzo 2013

8.2. Apéndice B: Herramientas de Desarrollo

Instalación de Eclipse, Herramientas de Edición Gráfica, SVN

Para poder editar el proyecto de OBO-Edit, hemos descargado los códigos abiertos desde el entorno de desarrollo integrado Eclipse[48] a través del Repositorio de Versiones SVN[49]. Mostramos a continuación cómo realizar la descarga del entorno de desarrollo y su correspondiente configuración.

La versión de Eclipse utilizada para el desarrollo de la tesina es *Eclipse Indigo SDK Version:* 4.2.0[50], utilizado en la distribución *Ubuntu* 12.04 *LTS* del sistema operativo Linux.

Una vez descargado el Eclipse, debemos incorporar la librería para poder editar las GUIs de OBO-Edit. Para ello abrimos Eclipse y nos dirigimos a `Help->Install New Software` en el menú. Allí vamos a la opción `Add` para añadir un repositorio de donde descargar software. Completamos como nombre: "Eclipse 3.8 WindowBuilder Repo" y como sitio web: `http://download.eclipse.org/windowbuilder/WB/integration/3.8`. Una vez cargado el sitio seleccionamos los paquetes: *Swing Designer*, *SWT Designer* y *WindowsBuilder Engine*.

Con estos paquetes cargados, cuando creemos código podremos hacer click derecho sobre el nombre de una clase gráfica y seleccionar la opción `Open With->WindowBuilder Editor` y se añadirá debajo de la ventana del código una solapa de nombre *Design* (además de la solapa *Source* que figuraba anteriormente). Si nos dirigimos a la nueva solapa tendremos todas las herramientas de interfaz gráfica para trabajar con ellas.

La siguiente figura ilustra cómo se ven las solapas:

[48] Plataforma típicamente usada para entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse. `http://www.eclipse.org/`, Fecha de consulta: Marzo 2013

[49] SVN es reconocido como un software de código abierto, centralizado en el control de versiones de forma segura. Es muy simple de usar y ayuda al desarrollo de todo tipo de proyectos. `http://subversion.apache.org/`, Fecha de consulta: Marzo 2013

[50] Para realizar los pasos de la descarga: `http://blog.rastersoft.com/?p=781`, Fecha de consulta: Marzo 2013

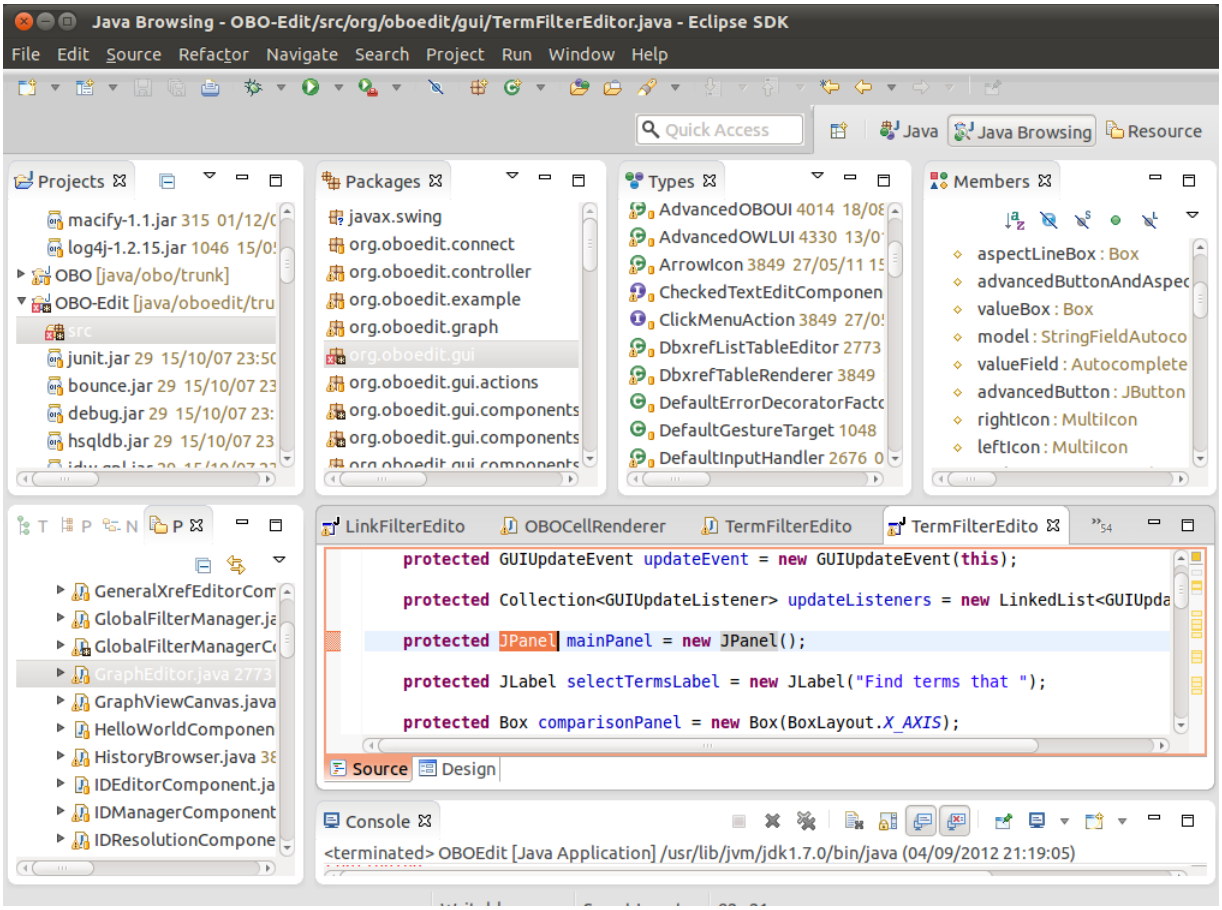

Figura 19: Vista de las dos solapas: Source y Design.

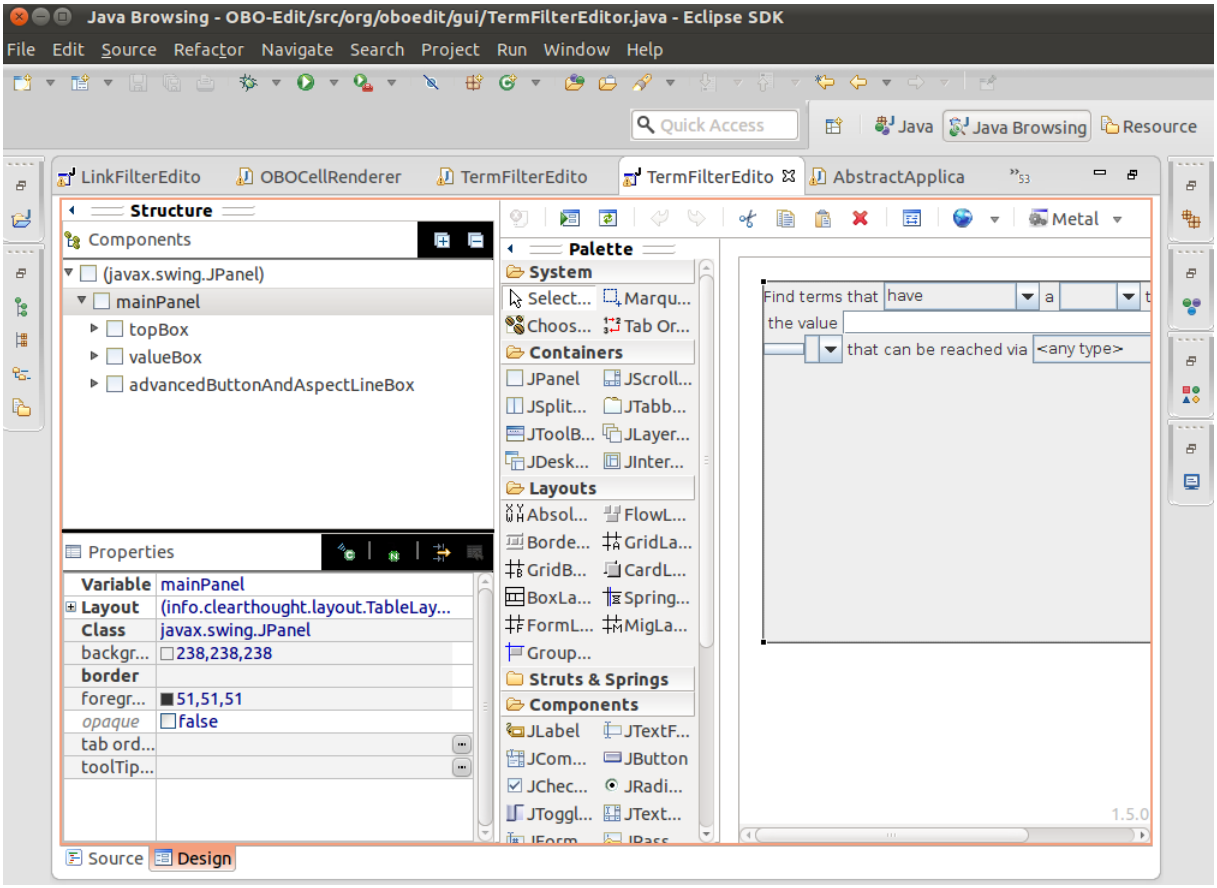

Figura 20: Vista de la solapa Design y las herramientas de diseño de interfaz gráfica.

De la misma manera podemos instalar el repositorio SVN yendo a la opción `Add` para añadir un repositorio y completando como nombre: "Subclipse 1.6.x (Eclipse 3.2+)" y como sitio web: `http://subclipse.tigris.org/update_1.6.x`. Una vez cargado el sitio seleccionamos el paquete: *Subclipse* y procedemos a la instalación[51].

[51] Para más información consultar: `http://subclipse.tigris.org/servlets/ProjectProcess?pageID=p4wYuA`, Fecha de consulta: Marzo 2013

8.3. Apéndice C: Códigos Fuente de OBO-Edit

Descarga de los Códigos de OBO-Edit con Eclipse y Ejecución del Programa

Luego de la instalación de Eclipse y SVN (ver Apéndice B (8.2)), procedemos a decargar los códigos de OBO-Edit[52] y sus APIs. Para ello nos dirigimos a `File ->New ->Project` y seleccionamos `Checkout Projects from SVN` en `New Project Wizard` y presionamos `Next`. Luego elegimos `Create a New Repository Location` y `Next`. Debemos completar con la dirección `https://geneontology.svn.sourceforge.net/svnroot/geneontology/java/oboedit/` y cliqueamos `Next`. Elegimos la carpeta `trunk` para obtener la última versión. Le damos un nombre al proyecto y cliqueamos `Finish`.

Debemos repetir el proceso para descargar las APIs pero con las siguientes direcciones: `https://geneontology.svn.sourceforge.net/svnroot/geneontology/java/obo/` para la API OBO y `https://geneontology.svn.sourceforge.net/svnroot/geneontology/java/bbop/` para la API BBOP.

Para poder correr el programa OBO-Edit desde Eclipse debemos dirigirnos a `Run` en la barra de menú y luego a `Run Configurations`. Como proyecto seleccionamos *OBO-Edit* y como clase principal elegimos *org.oboedit.launcher.OBOEdit*. A continuación cliqueamos `Run` y obtenemos nuestro editor de ontologías en ejecución.

[52] Para más información de cómo descargar los códigos, consultar: `http://wiki.geneontology.org/index.php/OBO-Edit:_Getting_the_Source_Code#Getting_OBO-Edit_from_Subclipse`, Fecha de consulta: Marzo 2013

8.4. Apéndice D: Filtro MF Manual

Realización de un Filtro MF en OBO-Edit de Forma Manual

Para comenzar a usar OBO-Edit, abrimos una ontología[53] desde la solapa `File` y la opción `Load Ontologies` de la barra de menú de OBO-Edit, y obtenemos:

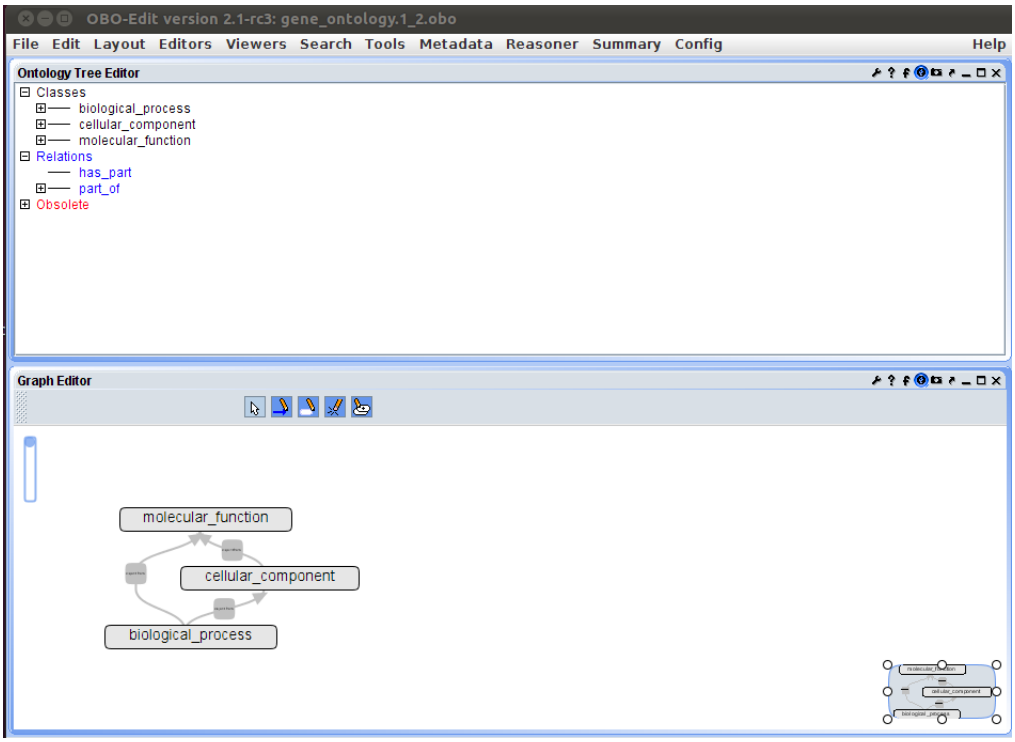

Figura 21: Vista de OBO-Edit al cargar una ontología.

En la parte titulada `Ontology Tree Editor` podemos observar las clases[54] y relaciones[55] de la ontología. Las tres clases que figuran corresponden a los tres dominios de la ontología GO: *biological_process*, *cellular_component* y *molecular_function*. Las relaciones existentes entre las clases de esta ontología particular son: *has_part* y *part_of*. En el recuadro titulado `Graph Editor` podemos ver el grafo correspondiente a la ontología, donde las flechas indican la relaciones entre sus clases. En este ejemplo vamos a realizar un filtro sobre GO (versión *OBO v1.2, Filtered ontology*) para quedarnos sólo con la clase *molecular function*. Para ello nos dirigimos a la solapa `File`, a

[53] Las ontologías pueden ser descargadas desde: `http://www.geneontology.org/GO.downloads.shtml`, Fecha de consulta: Agosto 2013

[54] También llamadas términos, son los conceptos descriptos por la ontología.

[55] Describen las relaciones entre los términos de la ontología.

la opción `Save As`, y luego `Advanced` para poder guardar la ontología nueva que estará filtrada[56]. Allí seleccionamos `Add` a la izquierda de la ventana. Para poder configurar los filtros seleccionamos las opciones `Filter terms`, `Filter links` y `Filter tags`.

Obtenemos el siguiente recuadro:

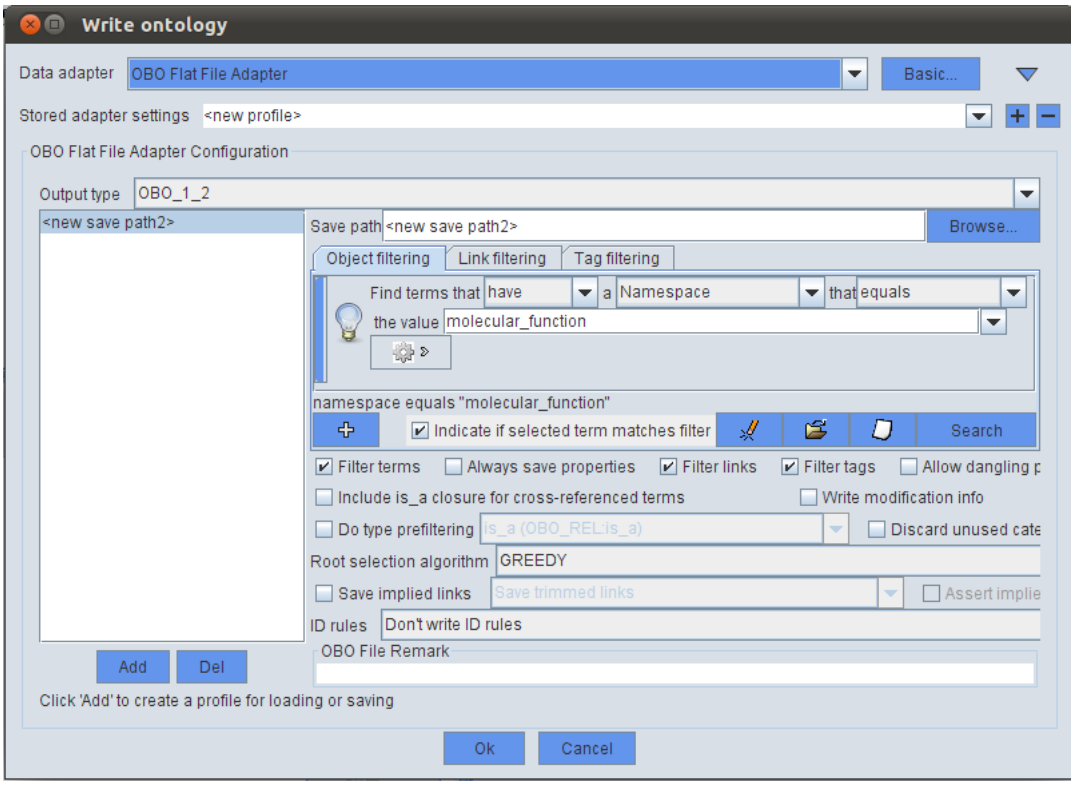

Figura 22: Vista de OBO-Edit con las opciones de filtrado.

Para realizar el filtro por la función molecular debemos tener en cuenta ciertos aspectos. La ontología *molecular function* tiene referencias a las ontologías *biological process* y *cellular component*, tanto en relaciones del tipo *molecular function* **part of** *biological process* como en las etiquetas *consider* y *replaced by* que recomiendan cómo reemplazar términos obsoletos en la ontología por otros. Para remover las relaciones *part of* anteriormente nombradas, se requiere un *link filter* y para remover las etiquetas se requiere un *tag filter*. Debemos aplicar cada filtro por separado:

- Object Filtering

  Se utiliza para realizar filtros sobre los objetos o clases de la ontología. En nuestro caso queremos aquellas clases que tengan como dominio

[56]Si aparecen Warnings, clickeamos en *Proceed*.

molecular function. Para ello armamos el filtro como: *Find terms that have a Namespace that equals the value molecular_function*

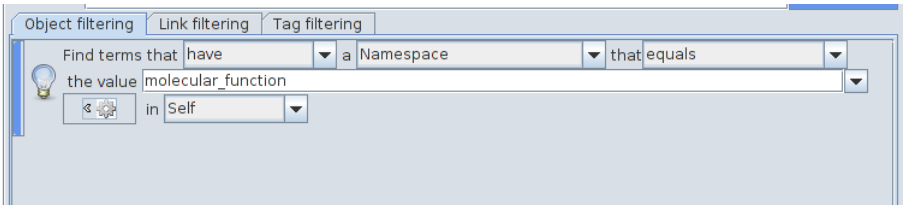

Figura 23: Object Filtering en OBO-Edit.

- Link[57] Filtering

  Se utiliza para realizar filtros sobre relaciones entre las clases de la ontología. En nuestro caso eliminamos la relación *part of*.

  Las relaciones se pueden eliminar de una ontología mirando el padre de la relación, el hijo, la relación en sí, o todo. En nuestro caso eliminamos las relaciones *part of* basándonos en todos los componentes de una relación, por eso utilizamos la opción *Type*.

  Para ello armamos el filtro como: *Find links where Type don't have a Name that contains the value part_of*

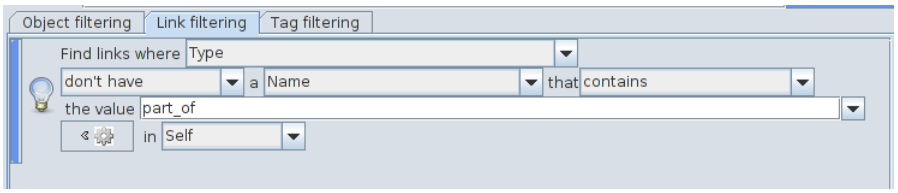

Figura 24: Link Filtering en OBO-Edit.

- Tag Filtering

  Este filtro habilita al usuario a elegir cuáles etiquetas serán guardadas en los archivos OBO.

  replaced_by devuelve un término que reemplaza un término obsoleto (en desuso). Sirve para reasignar automáticamente instancias cuya propiedad *instance_of* apunta a un término obsoleto.

  consider devuelve el término apropiado para sustituir un término obsoleto, pero necesita ser analizado por algún usuario experto antes de que ocurra la sustitución.

[57] Se denomina Link cuando dos términos se relacionan mutuamente.

No es necesario para nuestro filtro reemplazar los términos obsoletos así que desclickeamos las opciones *replaced_by* y *consider*

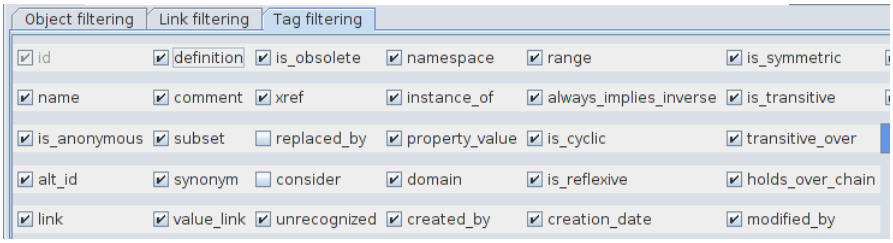

Figura 25: Tag Filtering en OBO-Edit.

Luego de setear los filtros, cliqueamos en el botón `Ok` y se genera un archivo con la ontología filtrada según el filtro especificado.

8.5. Apéndice E: Modificaciones Realizadas al Proyecto OBO-Edit

Códigos modificados y/o agregados al proyecto OBO-Edit. Los mismos se pueden consultar en la web: `https://www.dropbox.com/sh/q02sgt0o60mhci3/tDf7QqZSMA`.

Archivo:
`workspace/OBO-Edit/src/org/oboedit/automation/Filter_DB.java`

```
1  package org.oboedit.automation;
2  import java.io.BufferedReader;
3  import java.io.File;
4  import java.io.FileReader;
5  import java.io.FileWriter;
6  import java.util.ArrayList;
7
8  import javax.swing.JFileChooser;
9  import org.oboedit.automation.interfaces.DBFilter;
10
11 import org.apache.commons.csv.CSVParser;
12 import org.apache.commons.csv.CSVStrategy;
13
14 public class Filter_DB implements DBFilter {
15
16   File filterOntologyFile = new File("");
17   File ontologyFile = new File("");
18   File annotationFile = new File("");
19   boolean isFilteredOk = false;
20   ArrayList<String> ec_list;
21
22   public void setOntologyFile(String path)
23   {
24     ontologyFile = new File(path);
25
26   }
27
28   public void setFilterOntologyFile(String path)
29   {
30     filterOntologyFile = new File(path);
31
32   }
33
34   public void browse_ontology()
35   {
36     JFileChooser fc = new JFileChooser();
37         int respuesta = fc.showOpenDialog(fc);
38
39         if (respuesta == JFileChooser.APPROVE_OPTION)
40         {
41           ontologyFile = fc.getSelectedFile();
42         }
```

```
43
44    }
45
46    public void browse_filter_ontology()
47    {
48      JFileChooser fc = new JFileChooser();
49      int respuesta = fc.showOpenDialog(fc);
50
51          if (respuesta == JFileChooser.APPROVE_OPTION)
52          {
53              filterOntologyFile = fc.getSelectedFile();
54          }
55
56    }
57
58    public void browse_annotation()
59    {
60      JFileChooser fc = new JFileChooser();
61          int respuesta = fc.showOpenDialog(fc);
62
63          if (respuesta == JFileChooser.APPROVE_OPTION)
64          {
65              annotationFile = fc.getSelectedFile();
66          }
67
68    }
69
70    public void filterDatabase() {
71      try {
72
73        String ontologyPath = ontologyFile.getCanonicalPath();
74        String filterOntologyPath = filterOntologyFile.
              getCanonicalPath();
75        String annotationPath = annotationFile.getCanonicalPath();
76
77        String filterDBPathtemp = "../../filteredDBtemp.csv";
78        String filterDBPath = "../../filteredDB.csv";
79
80        File filterFile = new File(filterDBPathtemp);
81        filterFile.createNewFile();
82        File filterFileEC = new File(filterDBPath);
83        filterFileEC.createNewFile();
84
85        if (!filterFile.exists() || !filterFileEC.exists()) {
86          return;
87        }
88
89        String filterDBPathAbs = filterFile.getCanonicalPath();
90
91        String[] cmd = {"map2slim", filterOntologyPath,
              ontologyPath, annotationPath, "-o", filterDBPathAbs};
92
93        File wd = new File("../map2slim/go-perl-0.14/scripts");
94
```

```
95        Process proc = Runtime.getRuntime().exec(cmd,null,wd);
96          proc.waitFor();
97
98          FileReader fr = new FileReader(filterFile);
99        @SuppressWarnings("resource")
100       BufferedReader br = new BufferedReader(fr);
101
102       String linea = br.readLine();
103
104       if (linea != null) {
105         if (ec_list.size() == 0){
106           filterFile.renameTo(filterFileEC);
107         }
108         else {
109           //filtrar por evidence code
110           CSVParser parser = new CSVParser(new FileReader(
                filterDBPathtemp), CSVStrategy.EXCEL_STRATEGY);
111           @SuppressWarnings("resource")
112           FileWriter writer = new FileWriter(filterDBPath);
113           String temp = null;
114           String[] values = parser.getLine();
115           int i=0,j=0;
116           while (values != null) {
117             for(i=0;i<values.length;i++){
118               for (j=0; j < ec_list.size(); j++){
119                 temp="  "+ec_list.toArray()[j].toString()+" ";
120                 if (values[i].contains((CharSequence)temp))
121                   writer.write(values[i]+"\n");
122               }
123             }
124             values = parser.getLine();
125
126           }
127
128           filterFile.delete();
129         }
130         isFilteredOk = true;
131
132       }
133       else{
134         filterFile.delete();
135       }
136
137     } catch (Throwable e1) {
138       // TODO Auto-generated catch block
139       e1.printStackTrace();
140     }
141   }
142
143   public File getOntologyFile() {
144     return ontologyFile;
145   }
146
147   public File getFilterOntologyFile() {
```

```
148       return filterOntologyFile;
149     }
150
151     public File getAnnotationFile() {
152       return annotationFile;
153     }
154
155     public boolean isFiltered(){
156       return isFilteredOk;
157     }
158
159     public void setEvidenceCode(ArrayList<String> list){
160       ec_list = list;
161     }
162 }
```

Archivo:
workspace/OBO-Edit/src/org/oboedit/automation/Filter_MF.java

```
1  package org.oboedit.automation;
2  import java.util.ArrayList;
3  import java.util.Collection;
4  import java.util.HashSet;
5
6  import org.bbop.dataadapter.AdapterConfiguration;
7  import org.bbop.dataadapter.DataAdapter;
8  import org.bbop.framework.IOManager;
9  import org.obo.annotation.dataadapter.AnnotationParserExtension;
10 import org.obo.annotation.datamodel.AnnotationOntology;
11 import org.obo.dataadapter.OBOAdapter;
12 import org.obo.dataadapter.OBOConstants;
13 import org.obo.dataadapter.OBOFileAdapter;
14 import org.obo.dataadapter.OBOSerializationEngine;
15 import org.obo.dataadapter.OBO_1_2_Serializer;
16 import org.obo.dataadapter.OBOFileAdapter.
       OBOAdapterConfiguration;
17 import org.obo.dataadapter.OBOSerializationEngine.FilteredPath;
18 import org.obo.datamodel.IdentifiedObject;
19 import org.obo.datamodel.Link;
20 import org.obo.datamodel.OBOSession;
21 import org.obo.filters.Filter;
22 import org.obo.filters.TagFilter;
23 import org.oboedit.automation.interfaces.OntologyFilter;
24 import org.oboedit.controller.SessionManager;
25 import org.oboedit.gui.LinkFilterEditor;
26 import org.oboedit.gui.LinkFilterEditorFactory;
27 import org.oboedit.gui.TermFilterEditor;
28 import org.oboedit.gui.TermFilterEditorFactory;
29
30 public class Filter_MF implements OntologyFilter {
31     Collection<String> ontologyPath;
32
33   TagFilter tagFilter = new TagFilter();
34   HashSet<OBOConstants.TagMapping> tagsToWrite = new HashSet<
       OBOConstants.TagMapping>();
35
36   LinkFilterEditorFactory linkEF = new LinkFilterEditorFactory()
       ;
37   LinkFilterEditor linkEditor = new LinkFilterEditor();
38   Filter<Link> linkFilter;
39
40   TermFilterEditorFactory termEF = new TermFilterEditorFactory()
       ;
41   TermFilterEditor termEditor = new TermFilterEditor();
42   Filter<IdentifiedObject> termFilter;
43
44   boolean isFilteredOk = false;
45
46   public void filterOntology() {
```

```
47
48      //TAG FILTER
49      tagFilter.setReplaced_byTagToBeWritten(false);
50      tagFilter.setConsiderTagToBeWritten(false);
51      tagsToWrite = TagFilter.getTagsToWrite();
52
53      //LINK FILTER
54      linkEditor.layoutGUI_LinkFilterMF();
55      linkFilter = linkEF.getFilter(linkEditor);
56
57      //TERM FILTER
58      termEditor.layoutGUI_TermFilterMF();
59      termFilter = termEF.getFilter(termEditor);
60
61      try {
62        OBOFileAdapter adapter = new OBOFileAdapter();
63        OBOFileAdapter.OBOAdapterConfiguration config = new
              OBOFileAdapter.OBOAdapterConfiguration();
64
65        config.setBasicSave(false);
66        config.setAllowDangling(true);
67
68        OBOSession session = adapter.doOperation(OBOAdapter.
              WRITE_ONTOLOGY, config,
69            SessionManager.getManager().getSession());
70
71        session.importSession(AnnotationOntology.getSession(),
              true);
72        OBOSerializationEngine sengine = new
              OBOSerializationEngine();
73
74        FilteredPath filteredPaths = new FilteredPath();
75        filteredPaths.setObjectFilter(termFilter);
76        filteredPaths.setLinkFilter(linkFilter);
77        filteredPaths.setTagFilter(tagFilter);
78        filteredPaths.setTagsToWrite(tagsToWrite);
79        filteredPaths.setPath("../../filteredMF.obo");
80        filteredPaths.setDoFilter(true);
81        filteredPaths.setDoLinkFilter(true);
82        filteredPaths.setDoTagFilter(true);
83
84        Collection<FilteredPath> filteredPathCollection = new
              ArrayList<FilteredPath>();
85        filteredPathCollection.add(filteredPaths);
86
87        sengine.serialize(session, new OBO_1_2_Serializer(),
88            filteredPathCollection);
89        sengine.addSerializerExtension(new
              AnnotationParserExtension());
90
91            DataAdapter adapter2 = IOManager.getManager().
                  getCurrentAdapter();
92            AdapterConfiguration config2 = adapter2.
                  getConfiguration();
```

```
93
94              if (config2 instanceof OBOAdapterConfiguration) {
95                  OBOAdapterConfiguration oboconfig = (
                        OBOAdapterConfiguration)config2;
96                  ontologyPath = oboconfig.getReadPaths();
97              }
98
99          isFilteredOk=true;
100
101       } catch (Throwable e1) {
102         // TODO Auto-generated catch block
103         e1.printStackTrace();
104       }
105     }
106
107     public Filter<Link> getLinkFilter() {
108       return linkFilter;
109     }
110
111     public Filter<IdentifiedObject> getTermFilter() {
112       return termFilter;
113     }
114
115     public TagFilter getTagFilter() {
116       return tagFilter;
117     }
118
119     public HashSet<OBOConstants.TagMapping>  getTagsToWrite() {
120       return tagsToWrite;
121     }
122
123     public boolean isFiltered(){
124       return isFilteredOk;
125     }
126
127     public String getOntologyPath(){
128       return ontologyPath.toArray()[0].toString();
129     }
130 }
```

Archivo:
`workspace/OBO-Edit/src/org/oboedit/automation/interfaces/DBFilter.java`

```
1  package org.oboedit.automation.interfaces;
2  import java.io.File;
3  import java.util.ArrayList;
4
5  public interface DBFilter {
6
7     File getOntologyFile();
8     File getFilterOntologyFile();
9     File getAnnotationFile();
10
11    void filterDatabase();
12
13    void browse_ontology();
14
15    void browse_filter_ontology();
16
17    void browse_annotation();
18
19    public void setOntologyFile(String path);
20    public void setFilterOntologyFile(String path);
21
22    void setEvidenceCode(ArrayList<String> ec_list);
23
24    boolean isFiltered();
25
26 }
```

Archivo:
`workspace/OBO-Edit/src/org/oboedit/automation/interfaces/OntologyFilter.java`

```
1  package org.oboedit.automation.interfaces;
2  import java.util.HashSet;
3  import org.obo.dataadapter.OBOConstants;
4  import org.obo.datamodel.IdentifiedObject;
5  import org.obo.datamodel.Link;
6  import org.obo.filters.Filter;
7  import org.obo.filters.TagFilter;
8
9  public interface OntologyFilter {
10
11   void filterOntology();
12
13   Filter<Link> getLinkFilter();
14   Filter<IdentifiedObject> getTermFilter();
15   TagFilter getTagFilter();
16   HashSet<OBOConstants.TagMapping>  getTagsToWrite();
17
18   public String getOntologyPath();
19
20   boolean isFiltered();
21 }
```

Archivo:
workspace/OBO-Edit/src/org/oboedit/gui/LinkFilterEditor.java

```
1  public class LinkFilterEditor extends TermFilterEditor {
2
3  ///////////
4    public void layoutGUI_LinkFilterMF() {
5      if (aspectBox == null)
6        aspectBox = new JComboBox(values);
7      aspectBox.setSelectedIndex(1);
8      aspectBox.setEnabled(false);
9      if (selectLinkLabel == null)
10       selectLinkLabel = new JLabel("Find links where ");
11     Box northPanel = Box.createHorizontalBox();
12     northPanel.add(selectLinkLabel);
13     northPanel.add(aspectBox);
14     northPanel.add(Box.createHorizontalGlue());
15     notBox.setRenderer(new DefaultListCellRenderer() {
16       @Override
17       public Component getListCellRendererComponent(JList list,
18           Object value, int index, boolean isSelected,
19           boolean cellHasFocus) {
20         if (index == 0)
21           value = "has";
22         return super.getListCellRendererComponent(list, value,
                 index,
23               isSelected, cellHasFocus);
24       }
25     });
26     notBox.setSelectedIndex(1);
27     notBox.setEnabled(false);
28
29     //notBox.addActionListener(notBoxListener);
30     //aspectBox.addActionListener(aspectBoxListener);
31     comparisonBox.setSelectedIndex(0);
32     comparisonBox.setEnabled(false);
33     criterionBox.setSelectedIndex(2);
34     criterionBox.setEnabled(false);
35     valueField.setSelectedItem("part_of");
36     valueField.setEnabled(false);
37     typeBox.setSelectedIndex(0);
38     typeBox.setEnabled(false);
39
40 //     selectTermsLabel.setText("that");
41     selectTermsLabel.setText("");
42     setLayout(new BorderLayout());
43     add(northPanel, "North");
44     add(mainPanel, "Center");
45   }
46 ///////////
47
48 }
```

Archivo:
`workspace/OBO-Edit/src/org/oboedit/gui/TermFilterEditor.java`

```
1  public class TermFilterEditor extends JPanel {
2
3  //////////////
4    public void layoutGUI_TermFilterMF() {
5      notBox.setSelectedIndex(0);
6      notBox.setEnabled(false);
7
8      criterionBox.setSelectedIndex(8);
9      criterionBox.setEnabled(false);
10
11     valueField.setSelectedItem("molecular_function");
12     valueField.setEnabled(false);
13
14     comparisonBox.setSelectedIndex(1);
15     comparisonBox.setEnabled(false);
16
17     layoutGUI();
18   }
19 ///////////
20
21 }
```

Archivo:
workspace/OBO-Edit/src/org/oboedit/gui/automation/DBFilterGUI.java

```
1  package org.oboedit.gui.automation;
2  import java.awt.BorderLayout;
3  import java.awt.Color;
4  import java.awt.EventQueue;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import java.util.ArrayList;
8
9  import javax.swing.BorderFactory;
10 import javax.swing.JButton;
11 import javax.swing.JCheckBox;
12 import javax.swing.JComboBox;
13 import javax.swing.JFrame;
14 import javax.swing.JLabel;
15 import javax.swing.JOptionPane;
16 import javax.swing.JPanel;
17 import javax.swing.border.Border;
18 import javax.swing.border.EtchedBorder;
19 import javax.swing.border.TitledBorder;
20 import org.oboedit.automation.Filter_DB;
21 import org.oboedit.automation.interfaces.DBFilter;
22
23 public class DBFilterGUI {
24
25   JFrame frame;
26   DBFilter db_filter = new Filter_DB();
27
28   public JComboBox filterOntologyField = new JComboBox();
29   public JComboBox ontologyField = new JComboBox();
30   public JComboBox annotationField = new JComboBox();
31   JButton browseFilterOntology = new JButton("Browse Filter
         Ontology...");
32   JButton browseOntology = new JButton("Browse Ontology...");
33   JButton browseAnnotation = new JButton("Browse Annotation DB
         ...");
34   JButton acceptButton = new JButton("Ok");
35
36   JCheckBox evidenceIDA = new JCheckBox();
37   JCheckBox evidenceEXP = new JCheckBox();
38   JCheckBox evidenceIMP = new JCheckBox();
39   JCheckBox evidenceIEP = new JCheckBox();
40   JCheckBox evidenceISS = new JCheckBox();
41   /**
42    * Launch the application.
43    */
44   public void main() {
45     EventQueue.invokeLater(new Runnable() {
46       public void run() {
47         try {
48
```

```
49              frame.setVisible(true);
50
51              JPanel panel = new JPanel();
52              JPanel panel_south = new JPanel();
53
54              panel.setLayout(null);
55
56                panel.add(browseFilterOntology);
57                panel.add(filterOntologyField);
58                browseFilterOntology.setBounds(20,20,200,25);
59                filterOntologyField.setBounds(230,20,200,25);
60
61                panel.add(browseOntology);
62                panel.add(ontologyField);
63                browseOntology.setBounds(20,50,200,25);
64                ontologyField.setBounds(230,50,200,25);
65
66                panel.add(browseAnnotation);
67                panel.add(annotationField);
68                browseAnnotation.setBounds(20,80,200,25);
69                annotationField.setBounds(230,80,200,25);
70
71                JLabel label = new JLabel("Filter By Evidence Code:"
                      );
72                label.setBounds(20,120,200,25);
73
74                JLabel label_ida = new JLabel("IDA");
75                label_ida.setBounds(20,160,200,25);
76                JLabel label_exp = new JLabel("EXP");
77                label_exp.setBounds(20,180,200,25);
78                JLabel label_imp = new JLabel("IMP");
79                label_imp.setBounds(20,200,200,25);
80                JLabel label_iep = new JLabel("IEP");
81                label_iep.setBounds(20,220,200,25);
82                JLabel label_iss = new JLabel("ISS");
83                label_iss.setBounds(20,240,200,25);
84
85                evidenceIDA.setBounds(50,160,25,25);
86                evidenceEXP.setBounds(50,180,25,25);
87                evidenceIMP.setBounds(50,200,25,25);
88                evidenceIEP.setBounds(50,220,25,25);
89                evidenceISS.setBounds(50,240,25,25);
90
91                panel.add(label);
92                panel.add(label_ida);
93                panel.add(label_exp);
94                panel.add(label_imp);
95                panel.add(label_iep);
96                panel.add(label_iss);
97                panel.add(evidenceIDA);
98                panel.add(evidenceEXP);
99                panel.add(evidenceIMP);
100               panel.add(evidenceIEP);
101               panel.add(evidenceISS);
```

```
102             panel_south.add(acceptButton);
103
104             Border blackline = BorderFactory.createLineBorder(
                    Color.blue);
105             Border loweredetched = BorderFactory.
                    createEtchedBorder(EtchedBorder.LOWERED);
106             panel.setBorder(blackline);
107             panel.setBorder(loweredetched);
108             TitledBorder title;
109             title = BorderFactory.createTitledBorder("Select
                    Files");
110             panel.setBorder(title);
111
112             addlisteners();
113
114             frame.getContentPane().add(panel_south,BorderLayout.
                    SOUTH);
115
116             frame.add(panel);
117
118         } catch (Exception e) {
119           e.printStackTrace();
120         }
121       }
122     });
123   }
124
125   /**
126    * Create the application.
127    */
128   public DBFilterGUI() {
129     initialize();
130   }
131
132   /**
133    * Initialize the contents of the frame.
134    */
135   public void initialize() {
136     frame = new JFrame("Database Filter");
137     frame.setBounds(400, 400, 450, 350);
138     frame.setLocationRelativeTo(null);
139   }
140
141   public void start()
142   {
143     main();
144   }
145
146   public void addlisteners() {
147     browseFilterOntology.addActionListener(
            actionListener_load_filter_ontology);
148     browseOntology.addActionListener(
            actionListener_load_ontology);
149     browseAnnotation.addActionListener(
```

```
        actionListener_load_annotation);
150     acceptButton.addActionListener(actionListener_Filter);
151   }
152
153   ActionListener actionListener_load_ontology = new
          ActionListener() {
154     public void actionPerformed(ActionEvent e) {
155       try {
156         db_filter.browse_ontology();
157               ontologyField.addItem(db_filter.getOntologyFile().
                      getName());
158       } catch (Throwable e1) {
159         // TODO Auto-generated catch block
160         e1.printStackTrace();
161       }
162     }
163   };
164
165   ActionListener actionListener_load_filter_ontology = new
          ActionListener() {
166     public void actionPerformed(ActionEvent e) {
167       try {
168         db_filter.browse_filter_ontology();
169               filterOntologyField.addItem(db_filter.
                      getFilterOntologyFile().getName());
170       } catch (Throwable e1) {
171         // TODO Auto-generated catch block
172         e1.printStackTrace();
173       }
174     }
175   };
176
177   ActionListener actionListener_load_annotation = new
          ActionListener() {
178     public void actionPerformed(ActionEvent e) {
179       try {
180         db_filter.browse_annotation();
181         annotationField.addItem(db_filter.getAnnotationFile().
              getName());
182       } catch (Throwable e1) {
183         // TODO Auto-generated catch block
184         e1.printStackTrace();
185       }
186     }
187   };
188
189   ActionListener actionListener_Filter = new ActionListener() {
190     public void actionPerformed(ActionEvent e) {
191       try {
192
193         if (annotationField.getSelectedItem() == null ||
194              ontologyField.getSelectedItem() == null ||
195              filterOntologyField.getSelectedItem() == null) {
196            JOptionPane.showMessageDialog(null,
```

```
197                 "Please, complete the missing files",
198                 "Warning",
199                 JOptionPane.WARNING_MESSAGE);
200             return;
201         }
202
203         ArrayList<String> ec_list = new ArrayList<String>();
204
205         if (evidenceIDA.isSelected()){
206           ec_list.add("IDA");
207         }
208         if (evidenceEXP.isSelected()){
209           ec_list.add("EXP");
210         }
211         if (evidenceIMP.isSelected()){
212           ec_list.add("IMP");
213         }
214         if (evidenceIEP.isSelected()){
215           ec_list.add("IEP");
216         }
217         if (evidenceISS.isSelected()){
218           ec_list.add("ISS");
219         }
220
221         db_filter.setEvidenceCode(ec_list);
222         db_filter.filterDatabase();
223
224         if (db_filter.isFiltered()) {
225           JOptionPane.showMessageDialog(null,
226               "The databases have been filtered. See file
                      filteredDB.csv",
227               "",
228               JOptionPane.NO_OPTION);
229
230         }
231         else {
232            JOptionPane.showMessageDialog(null,
233                 "Can't filter the databases. Please, check if
                        the browsed files are correct or install
                        library GO:Perl",
234                 "Warning",
235                 JOptionPane.WARNING_MESSAGE);
236
237         }
238       } catch (Throwable e1) {
239         // TODO Auto-generated catch block
240         e1.printStackTrace();
241       }
242     }
243   };
244 }
```

Archivo:
workspace/OBO-Edit/src/org/oboedit/gui/automation/OntologyFilterGUI.java

```
1  package org.oboedit.gui.automation;
2  import java.awt.BorderLayout;
3  import java.awt.Color;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import java.io.IOException;
7  import javax.swing.BorderFactory;
8  import javax.swing.JButton;
9  import javax.swing.JCheckBox;
10 import javax.swing.JFrame;
11 import javax.swing.JLabel;
12 import javax.swing.JOptionPane;
13 import javax.swing.JPanel;
14 import javax.swing.border.Border;
15 import javax.swing.border.EtchedBorder;
16 import javax.swing.border.TitledBorder;
17 import org.bbop.dataadapter.DataAdapterException;
18 import org.oboedit.automation.Filter_MF;
19 import org.oboedit.automation.interfaces.OntologyFilter;
20
21 public class OntologyFilterGUI {
22
23   JFrame frame;
24
25   OntologyFilter mf_filter = new Filter_MF();
26
27   JCheckBox checkMF = new JCheckBox();
28   JButton acceptButton = new JButton("Ok");
29
30   /**
31    * Launch the application.
32    */
33   public void main() {
34     try {
35
36         frame.setVisible(true);
37
38         JPanel panel = new JPanel();
39           JPanel panel_south = new JPanel();
40           JLabel label = new JLabel("Molecular Function Filter")
                 ;
41
42           panel.add(label);
43           panel.add(checkMF);
44           panel_south.add(acceptButton);
45
46           addlisteners();
47
48           Border blackline = BorderFactory.createLineBorder(
                 Color.blue);
```

```
49            Border loweredetched = BorderFactory.
                  createEtchedBorder(EtchedBorder.LOWERED);
50            panel.setBorder(blackline);
51            panel.setBorder(loweredetched);
52            TitledBorder title;
53            title = BorderFactory.createTitledBorder("Select
                  Filter");
54            panel.setBorder(title);
55
56            frame.getContentPane().add(panel_south,BorderLayout.
                  SOUTH);
57            frame.add(panel);
58
59        } catch (Exception e) {
60          e.printStackTrace();
61        }
62      }
63
64    /**
65     * Create the application.
66     * @param string
67     * @throws IOException
68     * @throws DataAdapterException
69     */
70    public OntologyFilterGUI()  {
71      initialize();
72    }
73
74    /**
75     * Initialize the contents of the frame.
76     * @throws IOException
77     * @throws DataAdapterException
78     */
79    public void initialize()  {
80      frame = new JFrame("Ontology Filter");
81      frame.setBounds(400, 400, 300, 150);
82      frame.setLocationRelativeTo(null);
83    }
84
85    public void start()
86    {
87      main();
88    }
89
90    public void addlisteners() {
91      acceptButton.addActionListener(actionListener);
92    }
93
94    ActionListener actionListener = new ActionListener() {
95      public void actionPerformed(ActionEvent e) {
96        try {
97          if (checkMF.isSelected()) {
98            mf_filter.filterOntology();
99            if (mf_filter.isFiltered()) {
```

```
100                 JOptionPane.showMessageDialog(null,
101                     "The ontology have been filtered by Molecular
                            Function. See file filteredMF.obo",
102                     "",
103                     JOptionPane.NO_OPTION);
104
105             }
106
107             else {
108               JOptionPane.showMessageDialog(null,
109                     "Cant' filter the ontology. Please, try again",
110                     "Warning",
111                     JOptionPane.WARNING_MESSAGE);
112
113             }
114           }
115           else{
116             JOptionPane.showMessageDialog(null,
117                 "No filter have been selected",
118                 "Warning",
119                 JOptionPane.WARNING_MESSAGE);
120           }
121         } catch (Throwable e1) {
122           // TODO Auto-generated catch block
123           e1.printStackTrace();
124         }
125       }
126     };
127
128 }
```

Archivo:
workspace/OBO-Edit/src/org/oboedit/gui/automation/PipelineGUI.java

```
1  package org.oboedit.gui.automation;
2  import java.awt.BorderLayout;
3  import java.awt.Color;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import java.io.IOException;
7  import java.util.ArrayList;
8
9  import javax.swing.BorderFactory;
10 import javax.swing.JButton;
11 import javax.swing.JCheckBox;
12 import javax.swing.JFrame;
13 import javax.swing.JLabel;
14 import javax.swing.JOptionPane;
15 import javax.swing.JPanel;
16 import javax.swing.border.Border;
17 import javax.swing.border.EtchedBorder;
18 import javax.swing.border.TitledBorder;
19
20 import org.bbop.dataadapter.DataAdapterException;
21
22 public class PipelineGUI {
23
24   OntologyFilterGUI of_gui = new OntologyFilterGUI();
25   DBFilterGUI db_gui = new DBFilterGUI();
26
27   JFrame frame;
28   JCheckBox evidenceIDA = new JCheckBox();
29   JCheckBox evidenceEXP = new JCheckBox();
30   JCheckBox evidenceIMP = new JCheckBox();
31   JCheckBox evidenceIEP = new JCheckBox();
32   JCheckBox evidenceISS = new JCheckBox();
33   JButton acceptButton = new JButton("Ok");
34   /**
35    * Launch the application.
36    */
37   public void main() {
38     try {
39
40         frame.setVisible(true);
41
42         JPanel panel = new JPanel();
43         JPanel panel_south = new JPanel();
44
45         JLabel label = new JLabel("Choose Option For Ontology
               Filter:");
46           JLabel label1 = new JLabel("Molecular Function");
47           JLabel label2 = new JLabel("Choose Database:");
48           JLabel label3 = new JLabel("Filter By Evidence Code:")
                 ;
```

```
49
50              panel.setLayout(null);
51
52              panel.add(label);
53              label.setBounds(10,10,300,25);
54
55              panel.add(label1);
56              label1.setBounds(20,40,300,25);
57
58              panel.add(of_gui.checkMF);
59              of_gui.checkMF.setBounds(150,40,20,20);
60
61              addlisteners();
62
63              panel.add(label2);
64              label2.setBounds(10,70,400,25);
65
66              panel.add(db_gui.browseAnnotation);
67              panel.add(db_gui.annotationField);
68              db_gui.browseAnnotation.setBounds(20,100,200,25);
69              db_gui.annotationField.setBounds(230,100,200,25);
70
71
72              label3.setBounds(20,140,200,25);
73
74              JLabel label_ida = new JLabel("IDA");
75              label_ida.setBounds(20,180,200,25);
76              JLabel label_exp = new JLabel("EXP");
77              label_exp.setBounds(20,200,200,25);
78              JLabel label_imp = new JLabel("IMP");
79              label_imp.setBounds(20,220,200,25);
80              JLabel label_iep = new JLabel("IEP");
81              label_iep.setBounds(20,240,200,25);
82              JLabel label_iss = new JLabel("ISS");
83              label_iss.setBounds(20,260,200,25);
84
85              evidenceIDA.setBounds(50,180,25,25);
86              evidenceEXP.setBounds(50,200,25,25);
87              evidenceIMP.setBounds(50,220,25,25);
88              evidenceIEP.setBounds(50,240,25,25);
89              evidenceISS.setBounds(50,260,25,25);
90
91              panel.add(label3);
92              panel.add(label_ida);
93              panel.add(label_exp);
94              panel.add(label_imp);
95              panel.add(label_iep);
96              panel.add(label_iss);
97              panel.add(evidenceIDA);
98              panel.add(evidenceEXP);
99              panel.add(evidenceIMP);
100             panel.add(evidenceIEP);
101             panel.add(evidenceISS);
102
```

```
103             panel_south.add(acceptButton);
104             db_gui.addlisteners();
105
106             Border blackline = BorderFactory.createLineBorder(
                    Color.blue);
107             Border loweredetched = BorderFactory.
                    createEtchedBorder(EtchedBorder.LOWERED);
108             panel.setBorder(blackline);
109             panel.setBorder(loweredetched);
110             TitledBorder title;
111             title = BorderFactory.createTitledBorder("");
112             panel.setBorder(title);
113
114             frame.getContentPane().add(panel_south,BorderLayout.
                    SOUTH);
115             frame.add(panel);
116
117         } catch (Exception e) {
118           e.printStackTrace();
119         }
120       }
121
122     /**
123      * Create the application.
124      * @param string
125      * @throws IOException
126      * @throws DataAdapterException
127      */
128     public PipelineGUI()  {
129       initialize();
130     }
131
132     /**
133      * Initialize the contents of the frame.
134      * @throws IOException
135      * @throws DataAdapterException
136      */
137     public void initialize()  {
138       frame = new JFrame("Pipeline Filter");
139       frame.setBounds(400, 400, 450, 350);
140       frame.setLocationRelativeTo(null);
141     }
142
143     public void start()
144     {
145       main();
146     }
147
148     public void addlisteners() {
149       acceptButton.addActionListener(actionListener);
150     }
151
152     ActionListener actionListener = new ActionListener() {
153       public void actionPerformed(ActionEvent e) {
```

```
154         try {
155           if (of_gui.checkMF.isSelected()) {
156
157             if (db_gui.annotationField.getSelectedItem() == null)
                    {
158                JOptionPane.showMessageDialog(null,
159                     "Please, insert the annotation file",
160                     "Warning",
161                     JOptionPane.WARNING_MESSAGE);
162                return;
163             }
164
165             of_gui.mf_filter.filterOntology();
166             if (of_gui.mf_filter.isFiltered()) {
167               JOptionPane.showMessageDialog(null,
168                    "The ontology have been filtered by Molecular
                         Function. See file filteredMF.obo",
169                    "",
170                    JOptionPane.NO_OPTION);
171               try {
172
173                 db_gui.db_filter.setFilterOntologyFile("../../
                        filteredMF.obo");
174                 db_gui.db_filter.setOntologyFile(of_gui.mf_filter.
                        getOntologyPath());
175
176                 ArrayList<String> ec_list = new ArrayList<String
                        >();
177
178                 if (evidenceIDA.isSelected()){
179                   ec_list.add("IDA");
180                 }
181                 if (evidenceEXP.isSelected()){
182                   ec_list.add("EXP");
183                 }
184                 if (evidenceIMP.isSelected()){
185                   ec_list.add("IMP");
186                 }
187                 if (evidenceIEP.isSelected()){
188                   ec_list.add("IEP");
189                 }
190                 if (evidenceISS.isSelected()){
191                   ec_list.add("ISS");
192                 }
193
194                 db_gui.db_filter.setEvidenceCode(ec_list);
195
196                 db_gui.db_filter.setEvidenceCode(ec_list);
197                 db_gui.db_filter.filterDatabase();
198
199                 if (db_gui.db_filter.isFiltered()) {
200                   JOptionPane.showMessageDialog(null,
201                        "The databases have been filtered. See file
                             filteredDB.csv",
```

```
202                         "",
203                         JOptionPane.NO_OPTION);
204
205                 }
206                 else {
207                     JOptionPane.showMessageDialog(null,
208                             "Can't filter the databases. Please, check
                                 if the browsed files are correct or
                                install library GO:Perl",
209                             "Warning",
210                             JOptionPane.WARNING_MESSAGE);
211
212                 }
213               } catch (Throwable e1) {
214                 // TODO Auto-generated catch block
215                 e1.printStackTrace();
216               }
217             }
218
219
220             else {
221               JOptionPane.showMessageDialog(null,
222                   "Cant' filter the ontology. Please, try again",
223                   "Warning",
224                   JOptionPane.WARNING_MESSAGE);
225
226             }
227           }
228
229           else{
230             JOptionPane.showMessageDialog(null,
231                   "No filter have been selected",
232                   "Warning",
233                   JOptionPane.WARNING_MESSAGE);
234           }
235         } catch (Throwable e1) {
236           // TODO Auto-generated catch block
237           e1.printStackTrace();
238         }
239       }
240     };
241
242 }
```

Archivo:
workspace/OBO-Edit/src/org/oboedit/gui/menu/FileMenu.java

```
1  public class FileMenu extends DynamicMenu {
2
3  //////////////
4          protected static ActionListener
               filterOntologyActionListener;
5          protected static ActionListener
               filterDatabaseActionListener;
6          protected static ActionListener pipelineActionListener;
7  //////////////
8
9    public FileMenu() {
10
11 //////////////
12     JMenuItem filterOntologyItem = new JMenuItem("Ontology
           Filter");
13     JMenuItem filterDatabaseItem = new JMenuItem("Database
           Filter");
14     JMenuItem pipeline = new JMenuItem("Pipeline Filter");
15 //////////////
16
17 //////////////
18     add(filterOntologyItem);
19     add(filterDatabaseItem);
20     add(pipeline);
21 //////////////
22
23 //////////////
24     filterOntologyActionListener = new ActionListener() {
25       public void actionPerformed(ActionEvent e) {
26             doFilterOntology();
27       }
28     };
29     filterOntologyItem.addActionListener(
           filterOntologyActionListener);
30
31     filterDatabaseActionListener=new ActionListener() {
32       public void actionPerformed(ActionEvent e) {
33           doFilterDB();
34       }
35       };
36       filterDatabaseItem.addActionListener(
             filterDatabaseActionListener);
37
38       pipelineActionListener=new ActionListener() {
39       public void actionPerformed(ActionEvent e) {
40           pipeline();
41       }
42       };
43       pipeline.addActionListener(pipelineActionListener);
44 ///////////////////////
```

```
45
46    }
47
48 ////////////////
49        public static void doFilterOntology() {
50          try {
51            OntologyFilterGUI ontologyFilter = new
                  OntologyFilterGUI();
52            ontologyFilter.start();
53            } catch (Exception e1) {
54              // TODO Auto-generated catch block
55              e1.printStackTrace();
56              JOptionPane.showMessageDialog(null,
57                  "Can't filter the onlotogy.",
58                    "Warning",
59                    JOptionPane.WARNING_MESSAGE);
60            }
61          };
62
63          public static void doFilterDB() {
64            try {
65              DBFilterGUI databaseFilter = new DBFilterGUI();
66              databaseFilter.start();
67            } catch (Exception e1) {
68              // TODO Auto-generated catch block
69              e1.printStackTrace();
70              JOptionPane.showMessageDialog(null,
71                    "Can't filter the databases",
72                    "Warning",
73                    JOptionPane.WARNING_MESSAGE);
74            }
75          };
76
77          public static void pipeline() {
78            try {
79              PipelineGUI pipelineFilter = new PipelineGUI();
80              pipelineFilter.start();
81            } catch (Exception e1) {
82              // TODO Auto-generated catch block
83              e1.printStackTrace();
84              JOptionPane.showMessageDialog(null,
85                    "Can't do de pipeline filter",
86                    "Warning",
87                    JOptionPane.WARNING_MESSAGE);
88            }
89          };
90 ////////////
91
92 }
```

Bibliografía

Altman, R. B. (2006). *Guide to bioinformatics*. Stanford University. Descargado Agosto 2013, de `http://www-helix.stanford.edu/people/altman/bioinformatics.html`

Bajic, V. B., Brusic, V., Li, J., kiong Ng, S., y Wong, L. (2003). From informatics to bioinformatics. En *Proceedings of the first Asia–Pacific bioinformatics conference on bioinformatics, Adelaide, Australia* (p. 3-12). Descargado Agosto 2013, de `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.1237`

Biology-Online.org. (s.f.). *Biology-Online Dictionary*. Descargado Agosto 2013, de `http://www.biology-online.org/dictionary/`

Bonissone, P. (1997, Abril). Soft Computing: the Convergence of Emerging Reasoning Technologies. *Journal of Research in Soft Computing, Springer-Verlag*, *1*(1), 6-18.

Consortium, T. G. O., Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., ... Sherlock, G. (2011, Febrero). Gene Ontology: tool for the unification of biology. *PubMed Central Journal List*, *25*(1), 25-29. doi: 10.1038/75556

Crandall, K. A., y Lagergren, J. (2008). Algorithms in bioinformatics. *Lecture Notes in Computer Science*, *5251*, 15-19. doi: 10.1007/978-3-540-87361-7

Frasconi, P., y Shamir, R. (2003). *Artificial intelligence and heuristic methods in bioinformatics*. NATO Advanced Study Institute on Artificial Intelligence and Heuristics Methods in Bioinformatics: IOS Press.

Harris, M. (2011). *[go-helpdesk] GO MF and dangling references (from GO helpdesk)*. Geneontology-oboedit-working-group mailing list. E-mail: mah79 at cam.ac.uk. Descargado Agosto 2013, de `https://mailman.stanford.edu/pipermail/go-helpdesk/2011-December/004788.html`

Hill, D. P., Smith, B., McAndrews-Hill, M. S., y Blake, J. A. (2008). Gene Ontology annotations: what they mean and where they come from. *BMC Bioinformatics*, *9*(5), S2. doi: 10.1186/1471-2105-9-S5-S2

Ibba, M. (2002). Biochemistry and bioinformatics: when worlds collide. *Trends in Biochemical Sciences*, *27*(2). Descargado Agosto 2013, de `http://www.deepdyve.com/lp/elsevier/biochemistry-and-bioinformatics-when-worlds-collide-5LJNccP5iM`

Kanehisa, M., y Bork, P. (2003). Bioinformatics in the post-sequence era. *Nature Genetics*, *33*, 305-310. doi: 10.1038/ng1109

Lander, E. S., Michael S. Waterman, E. C. o. t. M. S. i. G., y Protein Structure Research, N. R. C. (1995). *Calculating the secrets of life:contributions of the mathematical sciences to molecular biology*. The National Academies Press. Descargado Agosto 2013, de `http://www.nap.edu/openbook.php?record_id=2121`

Lu, Z., y Hunter, L. (2005). GO Molecular Function Terms are Predictive of Subcellular Localization. *Pacific Symposium on Biocomputing*, *10*, 151-161. Descargado Marzo 2012, de http://helix-web.stanford.edu/psb05/lu.pdf

Medicina Molecular. (s.f.). *Medicina molecular glosario*. Descargado Agosto 2013, de http://www.medmol.es/glosario/89/

Murray-Rust, P., Mitchell, J. B. O., y Rzepa, H. S. (2005). *Chemistry in bioinformatics*. Descargado Marzo 2012, de http://www.dspace.cam.ac.uk/bitstream/1810/34580/1/1525766899692944_a_article1.pdf

Real Academia Española. (s.f.). *Diccionario de la lengua española* (22.ª ed.). Descargado Marzo 2013, de http://www.rae.es/rae.html

Stevens, R., Wroe, C., Lord, P., y Goble, C. (2004). *Ontologies in bioinformatics*. Department of Computer Science, University of Manchester, Oxford Road, Manchester UK, M13 9PL. Descargado Marzo 2012, de http://ias4.imise.uni-leipzig.de/ontomed_edit/Archiv/ontomed2002/de/lehre/ont-eng-2005-ss/protectedFiles/stevens-r-2004-635-a.pdf

Valentini, G., y Cesa-Bianchi, N. (2007). HCGene: a software tool to support the hierarchical classification of genes. *Bioinformatics*, *24*(5), 729-731. doi: 10.1093/bioinformatics/btn015

Woon, W. L. (2003). *Core statistics for bioinformatics*. Descargado Marzo 2012, de http://chagall.med.cornell.edu/BioinfoCourse/PDFs/Lecture3/bioinformatics_tutorial.pdf

Zadeh, L. A. (1994, Marzo). Fuzzy logic, neural networks and soft computing. *Communication of the ACM*, *37*(3), 77-84.

Glosario

A

ADN Biopolímero cuyas unidades son desoxirribonucleótidos y que constituye el material genético de las células y contiene en su secuencia la información para la síntesis de proteínas.

ARN Biopolímero cuyas unidades son ribonucleótidos. Según su función se dividen en mensajeros, ribosómicos y transferentes.

C

cromosoma Filamento condensado de ácido desoxirribonucleico, visible en el núcleo de las células durante la mitosis. Su número es constante para cada especie animal o vegetal.

célula Unidad fundamental de los organismos vivos, generalmente de tamaño microscópico, capaz de reproducción independiente y formada por un citoplasma y un núcleo rodeados por una membrana.

E

enzima Proteína que cataliza específicamente cada una de las reacciones bioquímicas del metabolismo.

G

gen Secuencia de ADN que constituye la unidad funcional para la transmisión de los caracteres hereditarios.

genoma Conjunto de los genes de un individuo o de una especie, contenido en un juego haploide de cromosomas.

L

locus (plural:loci) La ubicación de un gen (o de una secuencia significativa) en un cromosoma.

M

molécula Unidad mínima de una sustancia que conserva sus propiedades químicas. Puede estar formada por átomos iguales o diferentes.

N

núcleo Orgánulo celular limitado por una membrana y constituido esencialmente por cromatina, que regula el metabolismo, el crecimiento y la reproducción celulares.

O

organismo Conjunto de órganos del cuerpo animal o vegetal y de las leyes porque se rige.

órgano Cada una de las partes del cuerpo animal o vegetal que ejercen una función.

P

producto genético El material bioquímico, tanto ARN o proteína, resultante de la expresión de un gen. La cantidad de producto genético es usada para medir la actividad del gen, medidas anormales pueden relacionarse con alelos enfermos.

proteasoma Complejo macromolecular cuya función es la degradación de proteínas.

proteína Sustancia constitutiva de las células y de las materias vegetales y animales. Es un biopolímero formado por una o varias cadenas de aminoácidos, fundamental en la constitución y funcionamiento de la materia viva, como las enzimas, las hormonas, los anticuerpos, etc.

proteína dimérica Proteína formada por dos cadenas de aminoácidos.

R

retículo endoplasmático rugoso Orgánulo propio de la célula eucariota que participa en la síntesis y el transporte de proteínas.

ribosoma Orgánulo en el que tienen lugar las últimas etapas de la síntesis de proteínas.