BIOINFORMÁTICA:

CONSULTAS CRUZADAS A BASES DE DATOS BIOMÉDICAS REMOTAS

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Antonio Rodríguez Cía
i dirigit per
Mario Huerta
Juan José Villanueva
Bellaterra,.....de.....…....de 200...

Resumen:

En la presente memoria se detallan con exactitud los pasos y procesos realizados para construir una aplicación que posibilite el cruce de datos genéticos a partir de información contenida en bases de datos remotas.

Desarrolla un estudio en profundidad del contenido y estructura de las bases de datos remotas del NCBI y del KEGG, documentando una minería de datos con el objetivo de extraer de ellas la información necesaria para desarrollar la aplicación de cruce de datos genéticos.

Finalmente se establecen los programas, scripts y entornos gráficos que han sido implementados para la construcción y posterior puesta en marcha de la aplicación que proporciona la funcionalidad de cruce de la que es objeto este proyecto fin de carrera.

Resum:

En la present memòria es detallen amb exactitud els passos i processos realitzats per a construir una aplicació que possibiliti l'encreuament de dades genètiques a partir d'informació continguda en bases de dades remotes.

Desenvolupa un estudi en profunditat del contingut i estructura de les bases de dades remotes del NCBI i del KEGG, documentant una mineria de dades amb l'objectiu d'extreure d'elles la informació necessària per a desenvolupar l'aplicació d'encreuament de dades genètiques.

Finalment s'estableixen els programes, scripts i entorns gràfics que han estat implementats per a la construcció i posterior engegada de l'aplicació que proporciona la funcionalitat d'encreuament de la qual és objecte aquest projecte fi de carrera.

Summary:

This text describes the steps and processess that have been done in order to build an application that makes possible the genomic data crossing using information storaged in remote databases.

It develops an exhaustive study of NCBI's and KEGG's databases structure and content, documenting a data mining with the main purpose of extracting the necessary information from these databases, in order to build the genomic data crossing application.

It also describes the programs, scripts and grafical interfaces that have been used in order to create and run the final application that makes possible the main aim of this *End Of Career Purpose*.

Antonio Rodríguez Cía

ÍNDICE

0.- INTRODUCCIÓN:

El problema de determinar la función de una secuencia de DNA puede ser abordado desde diferentes vertientes, análisis de homología, búsqueda de motivos secuenciales, técnicas experimentales... etc.

Otra aproximación al problema puede ser la extracción de conocimiento a partir del análisis de los datos de microarrays. Para ello se analizan los niveles de expresión de los genes, obtenidos de técnicas masivas de obtención de datos usando microarrays. Gracias a ellas se pueden extraer los comportamientos relativos de los niveles de expresión de los diferentes genes. Esta información es muy importante ya que nos está determinando en cada momento cual es el conjunto de genes que se expresan. Pero los genes "per se" no suelen desempeñar función celular alguna, esa suele ser la función de las proteínas que se forman al expresarse los genes.

Conocer qué proteínas están interactuando en cada momento puede ser muy interesante ya que refuerza la idea de un función conjunta. Asociada al hecho de la expresión de los correspondientes genes (es decir, cómo operan y cuando son constituidas para poder operar).

Actualmente existen bases de datos especializadas en la que se almacena información sobre experimentos en los que se testa si dos proteínas interactúan, como pudieran ser las bases de datos de doble híbrido.

Después existen bases de datos sobre las publicaciones de estudios de proteínas y genes, donde se muestran y discuten el resultado de multitud de trabajos experimentales realizados en el laboratorio.

También existen bases de datos sobre vías de activación de genes (es decir, genes cuyas proteínas activan a otros genes para que se expresen éstos).

Los genes también han sido clasificados por su presunta función, presumible ubicación en la célula, etc...

Toda esta información asociada a cada gen puede ser accedida a través de las bases de datos biomédicas del **NCBI** (Nacional Center For Biotechnology Information http://www.ncbi.nlm.nih.gov/).

Teniendo pues toda esta información genética al alcance de la mano, sería de lo más interesante poder unificar en una aplicación un sistema de cruces que permita al usuario investigador obtener aquellos genes interesantes para el proceso de su investigación.

1.- OBJETIVOS:

El objetivo principal del proyecto será construir un cliente que consulte las bases de datos genéticas existentes en Internet, cruce los resultados según los parámetros de búsqueda indicados por el usuario y los muestre en un entorno fácilmente consultable.

La información cruzada será la relativa a las bases de datos Doble Híbrido, Pubmed, UniGene, AmiGO, KEGG, así como la información generada por nuestra herramienta para el análisis de la expresión génica obtenida por microarrays.

Principalmente se accederá a toda la información de genes a través de las Bases de datos del **NCBI**, accesibles mediante unas herramientas propias denominadas **eutils** *(Ver Anexo1)*.

El cliente de consultas habrá de añadirse al applet aplicación sobre tratamiento de microarrays ya existente en el servidor del departamento de Bioinformática. Tendremos por tanto una importante herramienta añadida a dicho applet, mejorando y enriqueciendo su funcionalidad.

El funcionamiento del cliente será sencillo, permitirá consultas encadenadas por **OR** o **AND** a las diferentes Bases de datos externas del **NCBI**. Dichas consultas podrán ir a su vez anidadas, siendo el resultado de una consulta la entrada a la siguiente.

Las entradas para las consultas serán los genes indicados por el usuario, los cuales se encuentre investigando y a partir de los que obtener nuevos genes según los parámetros de búsqueda que se indiquen en las consultas a las bases de datos externas.

Deberá verificarse que la entrada para una consulta sea correcta para la Base de datos correspondiente, así como optimizar las consultas encadenadas, tratando de obtener el resultado final en el menor tiempo posible.

Las salidas para dichas consultas serán todos aquellos genes que cumplan las condiciones impuestas, pudiéndose visualizar dicho resultado de forma gráfica en la aplicación ya existente y de forma detallada cómo una estructura jerárquica fácilmente navegable y legible vía web.

2.- CONCEPTOS PREVIOS:

2.1.- Bioinformática:

La Biología computacional o bioinformática es la ciencia dedicada al estudio de los fenómenos biológicos de la microbiología molecular desde un punto de vista computacional, con el objetivo de ofrecer métodos robustos para la comprensión, simulación y predicción de comportamientos biológicos observados en los seres vivos.

Bioinformática es un campo de la ciencia en el cual confluyen varias disciplinas tales como: biología, computación y tecnología de la información. El fin último de este campo es facilitar el descubrimiento de nuevas ideas biológicas así como crear perspectivas globales a partir de las cuales se puedan discernir principios unificadores en biología. Al comienzo de la "revolución genómica", el concepto de bioinformática se refería sólo a la creación y mantenimiento de base de datos donde se almacena información biológica, tales como secuencias de nucleótidos y aminoácidos. El desarrollo de este tipo de base de datos no solamente significaba el diseño de la misma sino también el desarrollo de interfaces complejas donde los investigadores pudieran acceder los datos existentes y suministrar o revisar datos.

La bioinformática consiguió parte de la importancia que tiene actualmente con el nacimiento del Proyecto Genoma Humano (HGP). No se trataba simplemente de leer la secuencia de DNA del ser humano, ya que la parte práctica del HGP (obtener la secuencia con métodos analíticos de laboratorio) ya hace varios años que finalizó, pero la parte teórica, comprender el significado de 3.2 GB de información, dista mucho de estar completada. Ante esta gran cantidad de datos que generan las nuevas técnicas de biología molecular, *el reto fundamental de la bioinformática es adaptar y utilizar técnicas computacionales existentes, o crear nuevas, para poder extraer información útil de estos datos (Data mining)*. Esta información ha de servir para complementar y validar la ya existente basada en experimentos clásicos, para generar nuevos conocimientos sobre determinados procesos biológicos y también descubrir nuevos.

Este es el principal fundamento de la realización de este proyecto. Intentaremos generar una nueva herramienta computacional que, a partir de una previa minería de datos realizada sobre la enorme cantidad de información existente sobre genes, obtenga aquella información útil para el entorno de aplicación subjetivo del usuario en cuestión.

Combinada con las nuevas técnicas de biotecnología, la bioinformática permite identificar genes y proteínas causantes de enfermedades o de mecanismos de resistencia a antibióticos, de otra forma complicados de detectar. En estos métodos, permite analizar de forma extensiva y lo suficientemente amplia los genes implicados, recreando un modelo aproximado, y en consecuencia pudiendo intervenir en los procesos con fármacos más específicos, o incluso con nuevos paradigmas terapéuticos, como la terapia génica.

La genómica, hasta hace poco la parte más importante de la bioinformática, consiste básicamente en el análisis de genomas, dedicada al descubrimiento y estudio de los genes que los componen. A partir de la secuenciación de un genoma la genómica interviene en diversos aspectos, uno de los principales es el ensamblaje de secuencias obtenidas de forma aleatoria para conseguir la secuencia completa y ordenada del genoma, y encontrar los genes que contiene.

Poco después del boom de la genómica, teniendo ya genomas con proteínas hipotéticas identificadas por métodos predictivos, hacía falta saber que proteína

expresaba cada secuencia. De aquí nació la proteómica, para descubrir qué es y qué función tiene una proteína. Este proceso se aplica hoy en día rutinariamente en lo que se conoce como anotación del genoma, es decir, dar nombre y función a todas las proteínas hipotéticas localizadas.

Un camino obvio hacia la proteómica es, simplemente, coger la secuencia de DNA, traducirla a proteína y predecir mediante simulación los plegamientos secundarios, terciarios y cuaternarios de la proteína y, en consecuencia, su estructura. Pero este camino presenta bastantes inconvenientes, por una parte se estima que, bien hecho, para una proteína pequeña (de unos 300 aminoácidos) se requieren 6 meses de proceso a 1 petaflop. Por otra parte, el modelo físico real es muy complejo y no se conoce lo suficiente como para poder simularlo en detalle. Así, todo y que se pudiera predecir bien el plegamiento a partir de cálculo, podría ser que se siguiese sin tener ni idea de que hace la proteína.

La bioinformática sería otro campo más de aplicación de técnicas computacionales si no fuera porque el campo sobre el que se aplican en este caso, la biología molecular, está experimentando un crecimiento exponencial. Con este panorama, el número y la complejidad de los datos aumentan a un ritmo más alto que la capacidad de los ordenadores que los han de procesar, la necesidad de un fuerte desarrollo teórico y práctico en el entorno de las herramientas bioinformáticas se hace más que necesario. Otro de los problemas a la hora de tratar toda esta información es la gran cantidad de ruido que hay, al tratarse de datos obtenidos experimentalmente.

Una vez se han clasificado los datos obtenidos, el último paso en el análisis de los mismos consiste en extraer la información biológica subyacente a los resultados experimentales. Para ello, se han desarrollado numerosas herramientas que obtienen información a partir de bases de datos u ontologías biológicas y permiten sacar conclusiones finales referentes al experimento realizado. Una especialidad dentro de la biotecnología es la denominada ingeniería metabólica, una nueva disciplina que tiene por objetivo último la optimización de las redes génicas, el estudio de la interacción entre los genes, así como el desarrollo de software para su estudio.

2.2.- Microarrays:

Para descubrir patrones de relaciones útiles en un conjunto de datos se empezaron a utilizar métodos que fueron denominados de diferente forma, con el término **Data Mining**. Desde este ángulo, del Data Mining aplica una dinámica que se mueve en sentido contrario al método científico tradicional. Con frecuencia, el investigador formula una hipótesis; luego, diseña un experimento para captar los datos necesarios y realiza los experimentos que confirmen o refuten la hipótesis planteada. Este es un proceso, que realizado de forma rigurosa, debe generar nuevos conocimientos. En el Data Mining, por el contrario, se captan y procesan los datos con la esperanza de que de ellos surja una hipótesis apropiada. Se desea que los datos nos describan o indiquen el por qué presentan determinada configuración y comportamiento.

El desarrollo clásico de un experimento de microarray de cDNA es un estudio de trascripción génica a lo largo del tiempo para dos condiciones, una será la de referencia y la otra será la propia del experimento. Así se podrá ver como afecta la condición dada a la trascripción de estos genes. Lo que se quiere ver son las diferencias en la expresión de los genes entre las dos condiciones. La trascripción

génica es el proceso de síntesis del RNA, que inicia la producción de las proteínas que determinan los procesos biológicos.

Un microarray de ADN consiste en un gran número de moléculas de ADN ordenadas sobre un sustrato sólido de manera que formen una matriz de secuencias en dos dimensiones. Estos fragmentos de material genético pueden ser secuencias cortas llamadas oligonucleótidos, o de mayor tamaño, cDNA (ADN complementario, sintetizado a partir de mRNA). A estos fragmentos de ADN de una sola hebra inmovilizados en un soporte, se les denomina a menudo sondas. Los ácidos nucleicos de las muestras a analizar se marcan por diversos métodos (enzimáticos, fluorescentes, etc.) y se incuban sobre el panel de sondas, permitiendo la hibridación (reconocimiento y unión entre moléculas complementarias) de secuencias homólogas. Durante la hibridación, las muestras de material genético marcadas, se unirán a sus complementarias inmovilizadas en el soporte del chip, permitiendo la identificación y cuantificación del ADN presente en la muestra (mutaciones, patógenos, etc.). Con posterioridad, el escáner y las herramientas informáticas permiten interpretar y analizar los datos obtenidos. Los microarrays de ADN, o micromatrices de material biológico, permiten la automatización simultánea de miles de ensayos.

En resumen, una microarray puede tratarse como una tabla donde las filas representan genes, las columnas experimentos y las celdas la expresión de dicho gen para dicho experimento.

2.2.1.- Tratamiento de Microarrays:

Hay una serie de de etapas en el tratamiento y estudio de los datos obtenidos con microarrays. Después de la obtención de los datos, éstos se normalizan y se realiza un análisis común.

Para facilitar el estudio de los datos obtenidos por microarrays se emplean clusters. El objetivo del análisis por clusters es, dada una matriz de genes y el resultado para diferentes experimentos realizados, encontrar un grupo de un conjunto de genes, de tal forma que los clusters resultantes sean homogéneos y estén bien separados. El punto clave es reducir la gran cantidad de datos caracterizándolos en grupos más pequeños de genes similares. Esto implica que los genes pertenecientes al mismo cluster han de ser tan similares entre ellos como sea posible, mientras que los genes de clusters diferentes han de ser tan diferentes como sea posible.

2.3.- Bases de datos del NCBI: [http://eutils.ncbi.nlm.nih.gov/]

El **Centro Nacional para la Información Biotecnológica** o **National Center for Biotechnology Information** (**NCBI**) es parte de la Biblioteca Nacional de Medicina de Estados Unidos, una rama de los Institutos Nacionales de Salud. Está localizado en Bethesda, Maryland y fue fundado el 4 de noviembre de 1988 con la misión de ser una importante fuente de información de biología molecular. Almacena y constantemente actualiza la información referente a secuencias genómicas en GenBank, un índice de artículos científicos referentes a biomedicina, biotecnología, bioquímica, genética y genómica en PubMed, una recopilación de enfermedades genéticas humanas en OMIM, además de otros datos biotecnológicos de relevancia en diversas bases de datos.

Todas las bases de datos del NCBI están disponibles en línea de manera gratuita, y pueden ser accesibles usando el buscador Entrez y sus herramientas Eutils *(Ver Anexo1)*.

De las bases de datos que proporciona el NCBI, para realizar nuestros cruces de genes nos interesarán aquellas que contengan la siguiente información relevante:

Bases de Datos del NCBI	
Base de datos	**Información**
Gene	Base de datos con información básica del gen: Taxonomy (Especie a la que pertenece), Símbolo, Nombre, Cromosoma en el que se encuentra, Interacciones con otros genes …
PubMed	Base de datos que contiene publicaciones en papers sobre experimentos e investigaciones en genes.
UniGene	Base de datos central cuya principal función es la unificación de los genes. Se empleará para la obtención de los GeneIDs (códigos de gen) que identifican unívocamente a cada gen.
OMIM	Base de datos que contiene patología de cada gen. Se empleará para cruzar genes por patologías.
Gene Ontology	Bases de datos con información acerca de la ontología de cada gen, de las cuales obtener que funciones realiza, en qué procesos intervienen y en qué componentes celulares se localizan.
Homologene	Base de datos con información acerca de genes homólogos. El empleo de esta información no será más que para descartar aquellos genes resultado que sean homólogos.

2.3.1.- Bases de datos del KEGG: [http://www.genome.jp/kegg/]

Como base de datos auxiliar, emplearemos la KEGG (**Kyoto Encyclopedia of Genes and Genomes**) de la que emplearemos principalmente los **KEGG PATHWAYS.** Estos pathways son una colección de mapas dibujados manualmente en los que se representan interacciones moleculares y redes de reacción para:

- Metabolismos
- Información de Procesos Genéticos
- Información de Procesos del Entorno del gen
- Procesos celulares
- Enfermedades humanas

Ver más información en *(5.2.7.- Base de datos KEGG (Pag 85))*

3.- ESTUDIO DE VIABILIDAD:

Para el estudio de viabilidad del proyecto, analizaremos y describiremos inicialmente el sistema propuesto y veremos la posibilidad de elegir entre diferentes soluciones posibles al problema. Una vez decidida la opción a implementar, se realizará un análisis de costes y de tamaño, para luego definir cualitativa y cuantitativamente los beneficios del mismo.

3.1.- Análisis y descripción del sistema propuesto:

3.1.1.- Applet de consultas online:

Como definimos anteriormente, nuestro sistema en última instancia, será una herramienta incorporada al applet ya existente en el servidor del departamento de Bioinformática de la UAB. Este applet es un motor gráfico que dispone una microarray de genes en un entorno 2D, organizándolos por cluster y por grado de correlación entre ellos.

A partir de esta disposición gráfica de genes, el usuario puede manipularlos, obteniendo información acerca de ellos y de sus relaciones. Esta tecnología visual basada en las librerías Java **JUNG** *(ver Anexo2)* permite el tratamiento de microarrays de forma sencilla e intuitiva, siendo un marco perfecto para la labor del investigador.

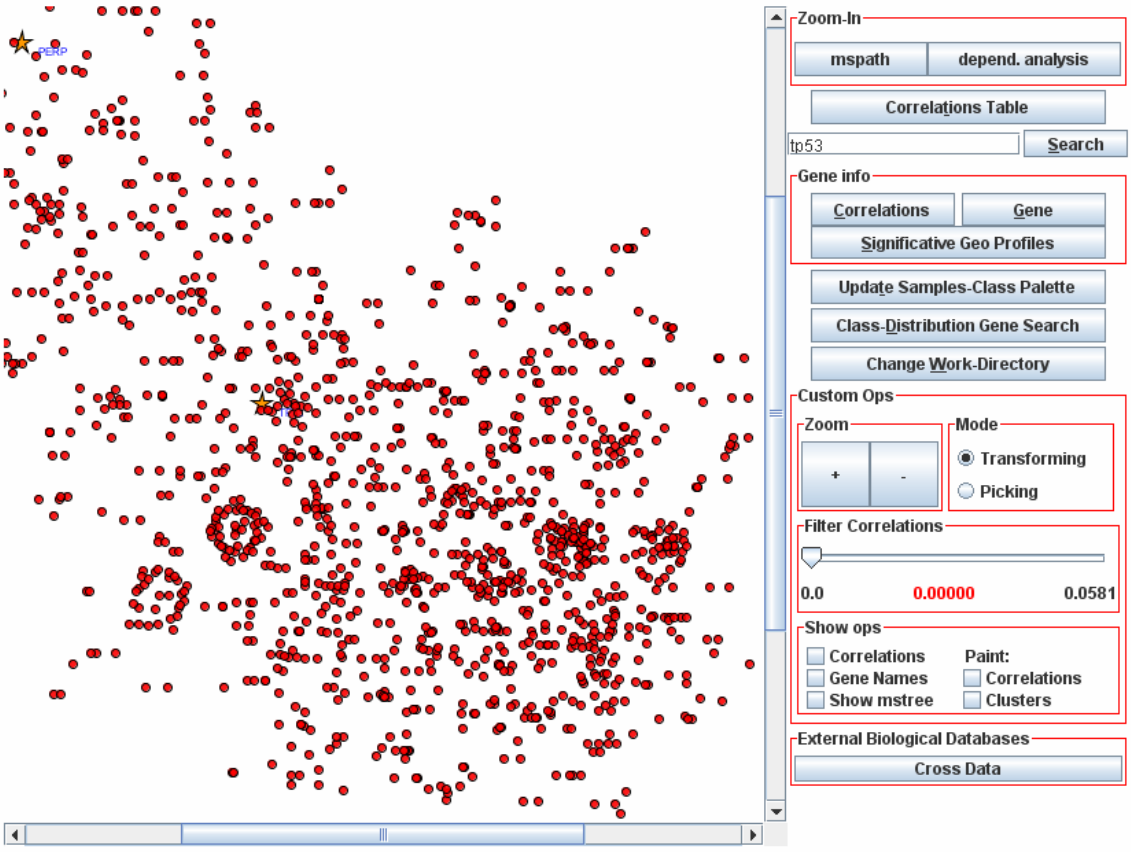
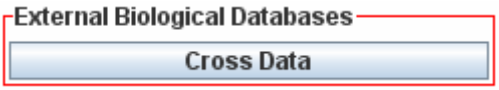


Imagen3.1.1: Aplicación Bio Project del IBB
[http://revolutionresearch.uab.es/applic/gexp/microarray/BioProject.html]

El objetivo de este proyecto no es más que dotar de funcionalidad al botón que aparece en última instancia en el menú de la derecha de la aplicación:



Así, dado una serie de genes orígenes que se marcarán en el grafo de la microarray, el usuario podrá cruzar información acerca de ellos, accediendo para esto a las bases de datos externas que se encuentran almacenadas en el NCBI *(Ver apartado 2.3)*.

En un posterior análisis detallado del diseño lógico de las bases de datos a utilizar, veremos qué datos en concreto se empleará para realizar el cruce de información entre genes.

Las consultas a realizar a estas bases de datos externas tendrán el siguiente formato:

- Genes orígenes seleccionados en la aplicación.
  - Realizar un AND u OR de los genes orígenes:
    - OR: mostrar genes solución para cada gen origen.
    - AND: mostrar genes solución comunes para todos los genes orígenes.
- Bases de datos sobre las que cruzar información (PubMed, KEGG, OMIM, Interactions, GO, Locations…).
- Parámetros para filtrar búsquedas en cada base de datos:
  - Indicar nombre de Function, Component o Process que deseamos para los genes resultado (Ej: genes que aparezcan en Función *ATP Binding*. Genes que aparezcan en Componente *Cell Nucleus*).
  - Indicar clave para obtener paper de PubMed (Ej: genes que aparezcan en papers que traten sobre *Cancer*).
- Dominios de búsqueda sobre los que obtener un conjunto de genes solución:
  - Genes seleccionados (picados en el grafo de la microarray y que son a su vez origen de la consulta).
  - Genes del mismo cluster que los seleccionados.
  - Genes con determinado grado de correlación que los seleccionados.
  - Genes en la microarray.
  - Todos los genes. Para este último caso, sólo se pintarán en el grafo como solución los genes que estén en la microarray actual. El resto de genes aparecerán en la página web de resultados.

El formato de la salida se dividirá en dos:

- Por una parte generaremos una lista de genes, ordenados jerárquicamente según las Bases de datos elegidas para cruzar información. Esta lista será una página web sobre la que se podrá obtener cualquier tipo de información relacionada con los genes obtenidos como resultado. Es decir, enlaces a página de cada gen en la web de NCBI, a páginas de papers de PubMeds, Ontologías, Funciones, Componentes, Procesos, mapas genéticos del KEGG…
- Por otra parte, los genes solución se marcarán en el grafo de la aplicación, sirviendo éstos para una futura entrada de una nueva tarea en el proceso. Se hará distinción a la hora de marcar entre los genes que han sido origen de la consulta y aquellos que se han obtenido como resultado.

El proceso explicado hasta ahora es el programa que ha de obtenerse como resultado final, agregando su funcionalidad al applet existente enriqueciendo así la herramienta.

Previamente a la realización de la herramienta de cruces online sobre la microarray, será necesaria la creación de un preproceso cuyo principal objetivo es obtener los nombres de genes actualizados.

3.1.2.- Preproceso para la obtención de nombres de gen:

Las bases de datos del NCBI se encuentran en continua actualización, ya que cada día surgen nuevos experimentos y se obtienen nuevos resultados que pasan a engrosar sus bases de datos.

Es por esto que el estudio de genes no es algo cerrado, finito y terminado. Al contrario, poco a poco se va adquiriendo más y más conocimiento acerca de cada gen, cómo se expresan, dónde aparecen, cual es su función etc.

Tenemos pues que subirnos a esta vorágine de información y mantenernos actualizados en cada momento.
En nuestra aplicación, cada vez que un usuario sube una microarray, se deberá consultar a las bases de datos del NCBI por el nombre y código actual de cada gen contenido en la microarray.

Para realizar esta labor, será necesario construir un robot de preproceso, de manera que correrá en background al generarse una microarray. Su principal labor será el obtener el nombre de gen actual para cada gen de la microarray, partiendo de los identificadores contenidos en la propia microarray.

Este programa será el encargado de generar los ficheros con los nombres de los genes de la microarray. Estos ficheros serán los que posteriormente utilizará el entorno gráfico del applet para generar el grafo en 2D de la microarray en cuestión.

El primer paso en nuestro proyecto será la creación de este preproceso de obtención de nombres de gen, ya que sin los ficheros que genera no tiene sentido construir el applet de consultas on-line.
Una vez que el preproceso sea correcto y las microarrays se encuentren debidamente actualizadas, se procederá a la realización de consultas on-line a las bases de datos externas del NCBI, trabajando a partir de este momento con los nombres de gen debidamente actualizados.

3.2.- Definición y coste de posibles soluciones al problema propuesto:

3.2.1.- Solución inicial: Consultas a bases de datos remotas usando Eutils:

La solución trivial que a priori resuelve nuestro problema planteado es el empleo de las herramientas remotas que proporciona la página web del NCBI. Estas herramientas son una serie de consultas html que devuelven resultados en diferentes formatos (xml,txt,html) sobre cualquier información contenida en las bases de datos necesarias para realizar los cruces de genes. Son las denominadas **eutils** *(ver Anexo1).*

Así pues, partiendo de los genes orígenes, empleando las eutils podríamos obtener sin problema (tras realizar los debidos cruces enlazados) la lista de genes resultado para una consulta determinada.

El coste de esta solución en espacio sería casi nulo, ya que no es necesario mantener una base de datos local puesto que toda la información se encuentra centralizada y actualizada en las bases de datos americanas del NCBI.

El principal problema de este sistema on-line empleando *eutils* es que el servidor de NCBI sólo permite un determinado número de consultas al minuto y si se supera este número, la conexión se cierra durante un periodo de tiempo.

Este inconveniente es vital, puesto que el coste temporal para consultas que hayan de tener en cuenta un gran número de genes puede dispararse y tardar del orden de minutos. Además, puesto que la aplicación estará situada en el servidor del IBB, el acceso por parte de muchos usuarios podría generar un número ingente de consultas web al NCBI, con el correspondiente bloqueo de la máquina que realiza dichas consultas.

En este peor caso, el resultado es inadmisible ya que la máquina servidora que contiene la aplicación se vería privada de acceso a las eutils y la funcionalidad de la aplicación estaría *de baja* durante un periodo de tiempo que variará según disponga la central del NCBI.

Nuestro principal deseo es que la aplicación de consultas on-line esté disponible en cualquier momento y una pérdida de la funcionalidad por causas de un agente ajeno no es admisible.

Será necesario buscar una alternativa a esta opción.

3.2.2.- Solución final: Construcción de una base de datos local:

Para solucionar este inconveniente, sería interesante disponer en una base de datos local la información necesaria para poder desarrollar la aplicación de generación de consultas en un formato adecuado.

El NCBI proporciona también de manera gratuita acceso a su sitio FTP [ftp://ftp.ncbi.nih.gov/], en el cual se alojan todos los ficheros necesarios para construir la totalidad de bases de datos genéticas que existen.

Estos ficheros se actualizan diariamente y conforman la base de conocimiento que existe actualmente en cada momento sobre experimentos e investigaciones que involucran genes.

Una posible nueva vía alternativa a la solución proporcionada en el apartado anterior es efectuar un *Data Mining* de los datos existentes en el FTP del NCBI. Así, sería posible construir en el servidor local del IBB una base de datos que contenga toda la información relevante y necesaria para realizar la aplicación de consultas on-line.

Todas aquellas consultas complejas que mediante eutils puedan bloquear el acceso al servidor, podrán ser llevadas a cabo consultando la base de datos local.

Con esta solución se consigue reducir el coste temporal de la futura aplicación, ya que las consultas complejas se harán sobre una base de datos local, siendo el tiempo de respuesta de ésta del orden de los segundos.

El principal problema de esta nueva alternativa surge de la necesidad de construir una base de datos local. En el anterior caso, el coste espacial era nulo puesto que toda la información se accedía de forma remota.
En esta solución, el coste espacial será del orden de los GigaBytes ya que la base de datos a construir en local contendrá toda la información pertinente y necesaria sobre todos los genes conocidos hasta el momento.

Además, añadir que habrá que diseñar un nuevo robot de preproceso encargado de generar y actualizar cuando se requiera la base de datos local, en el sentido de que la información local sea coherente con la existente en las bases de datos remotas del NCBI

Dicho robot de preproceso se conectará automáticamente al FTP del servidor del NCBI, se bajará los ficheros que contengan información útil y que han sido actualizados desde el último acceso por parte del robot. Posteriormente, otro robot usará estos archivos para obtener de ellos la información necesaria y actualizará el contenido de las bases de datos locales.

Así pues dispondremos de un sistema automático que se descargará la última versión de los datos de genes recopilados por el NCBI, y los almacenará en nuestra base de datos local para ser tratados finalmente por la aplicación de consultas on-line.

El acceso a los datos genéticos en esta alternativa por tanto se realiza localmente, excepto algunas consultas sencillas que seguirán realizándose mediante el empleo de las *eutils.*

3.2.3.- Comparación de alternativas:

Vemos en la siguiente tabla un resumen de las conclusiones alcanzadas en los apartados anteriores.

<table>
<tr><th colspan="2">Coste Temporal</th></tr>
<tr><th>Versión Inicial</th><th>Versión Final</th></tr>
<tr><td>Del orden de los minutos.
Ante una gran demanda de información, posibilidad de bloqueo de acceso a los datos y pérdida de la funcionalidad de la aplicación.

¡Inadmisible!</td><td>Del orden de los segundos.
Acceso a la información de manera rápida y eficiente haciendo uso de una base de datos local previamente actualizada.</td></tr>
<tr><th colspan="2">Coste Espacial</th></tr>
<tr><th>Versión Inicial</th><th>Versión Final</th></tr>
<tr><td>Nulo.
Los datos necesarios para la aplicación de consultas on-line se encuentran almacenados de forma remota en las bases de datos del NCBI.</td><td>Del orden de los GigaBytes.
La base de datos local contendrá toda la información necesaria para evitar consultas mediante eutils.</td></tr>
<tr><td></td><td>Requiere la implementación adicional de una serie de robots que conecten con el FTP del NCBI, descarguen ficheros actualizados, parseen su contenido y actualicen las bases de datos locales.</td></tr>
</table>

La versión que se llevará a cabo para la consecución del proyecto será por tanto la final.

Vemos que la elección de esta alternativa nos obliga a construir un robot que, cada cierto tiempo *(ver Punto 8 Distribución),* actualizará los datos locales para que estén en concordancia con los datos remotos oficiales.

Esta actualización consumirá recursos de la máquina dónde se implementará la base de datos local.
Durante el tiempo que dicho preproceso se encuentre actualizando la base de datos, no será necesario desactivar la aplicación puesto que el sistema de gestión de base de datos nos garantiza un acceso concurrente y en exclusión mutua a los datos.

El único inconveniente será que los resultados obtenidos en las consultas realizadas durante el periodo de actualización puede que no sean completos. Preferimos que los resultados obtenidos sean un subgrupo de un posible grupo real de resultados a tener inaccesible la aplicación durante este periodo de tiempo.

3.3.- Definición del marco temporal:

3.3.1.- Planificación de tareas:

A continuación se muestra una planificación temporal del proceso a desarrollar. Se ha tenido en cuenta que la duración del proyecto es de 9 meses, comenzando en Octubre y finalizando en Junio.

Debemos tener en cuenta que, cuando hablamos de procesos de desarrollo, no es suficiente con pensar en un proceso lineal de guionización, creación de los elementos y desarrollo de software, sino que hay que imaginar un proceso algo más complejo.

La mayoría de proyectos deben realizarse en diversas etapas. Algunas etapas comenzarán cuando otras etapas anteriores concluyan, aunque habrá etapas que podrán solaparse y realizarse al mismo tiempo.

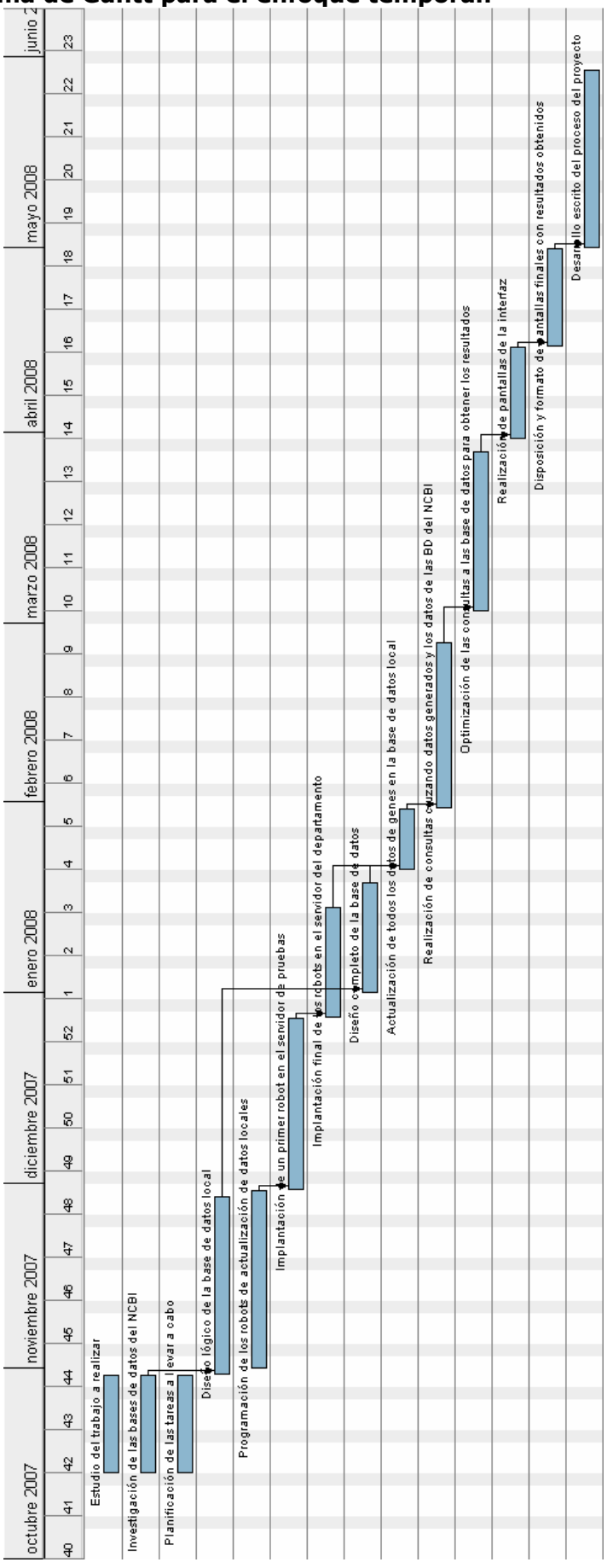
Así pues, el proceso de desarrollo del software es un proceso cíclico, que requiere de retroalimentación entre diversas etapas, estando todas ellas interrelacionadas y pudiendo volver atrás para modificar cualquier anomalía en un determinado momento.

Tener en cuenta que el hecho de *volver atrás* será más costoso a medida que avancemos en el proyecto, ya que modificar tareas anteriores afectará negativamente a todo el proceso posterior llevado a cabo.

Octubre 2007	·Estudio del trabajo a realizar. ·Investigación de las bases de datos del NCBI. ·Planificación temporal de las tareas a llevar a cabo.
Noviembre 2007 **Diciembre 2007**	·Programación de los robots de actualización de datos locales a partir de Data Mining realizado sobre BD's del NCBI. ·Diseño lógico de la base de datos local. ·Implantación de un primer robot en un servidor de pruebas.
Enero 2008	·Implantación final de los robots de actualización de datos en el servidor del departamento. ·Diseño completo de la base de datos local. ·Actualización de todos los datos remotos de genes en la base de datos local.
Febrero 2008 **Marzo 2008**	·Realización de consultas cruzando datos generados y datos de las BD del NCBI. ·Optimización de las consultas a las base de datos para obtener los resultados de los cruces.
Abril 2008	·Realización de pantallas de la interfaz del applet. ·Disposición, integración y formato de pantallas finales en aplicación existente, mostrando resultados obtenidos.
Mayo 2008	·Desarrollo escrito del proceso del proyecto, requisitos iniciales, análisis, diseño, implementación, mantenimiento, métrica empleada, etc …
Junio 2008	·Lectura del proyecto

Veamos a continuación un diagrama Gantt con las tareas en un gráfico temporal, así como las dependencias entre ellas:

3.3.2.- Diagrama de Gantt para el enfoque temporal:



3.4.- Recursos necesarios para el desarrollo del software:

3.4.1.- Equipo de trabajo:

El desarrollo de un proyecto requiere la implicación de un equipo de trabajo en el que cada miembro desempeña su función individualmente, para luego conformar el proyecto final a partir del trabajo realizado.

Cada miembro del proyecto dispondrá de unas habilidades específicas para realizar cada tarea.

El número de personas que conformarán este equipo dependerá de la magnitud del proyecto a desarrollar.
En general se requiere personal que cubra las siguientes funciones:

Todo proyecto necesitará de un **gerente** o **director**, responsable del desarrollo total e implementación del mismo, manejando presupuestos, horarios, programación de tareas y colaborando con cada grupo de trabajo.

El **responsable de contenido** será el encargado de elaborar un documento que contenga toda la información de carácter relevante para el proyecto. Además, proporcionará a cada grupo de trabajo la información necesaria en cada momento, sintetizada, redactada y comunicada de forma clara y concisa.

Si el proyecto los requiere, serán necesarios los **expertos en medios** (imagen fija, imagen en movimiento, video, modelado 3D, animación 3D) que desarrollarán su labor de acuerdo a los requisitos del propio proyecto.

El **diseñador multimedia** será el encargado del contenido global del proyecto, establecer una estructura básica para el contenido, determinar qué elementos de diseño son necesarios para soportar dicha estructura, establecer los métodos para presentar el contenido y, en menor medida, actuar de educador y de medio de difusión del producto.

Además, el diseñador multimedia actuará de diseñador gráfico, creador de interfaces y desarrollador de los elementos interactivos que, en pantalla, interactuarán con el usuario.

Los **programadores** de la aplicación serán los encargados de integrar todos los elementos previamente desarrollados y dar forma al proyecto final. El desarrollo se lleva a cabo con la ayuda de uno o varios lenguajes de programación que se ajusten a las necesidades del proyecto.

Observamos por tanto que son necesarias muchas habilidades para elaborar un producto que satisfaga todos los requisitos. Es necesario dividir el proyecto en tareas y asignar a cada una un equipo de trabajo que la resuelva de la mejor manera posible, siempre bajo la supervisión del gerente o director.

El cálculo apropiado de cuánto personal se requiere está en función de la cantidad de recursos a preparar digitalmente, de la complejidad de los mismos y del tipo de interacciones (simples o complejas) que se incluirán.

En este proyecto, yo he tomado el papel de todas las funciones anteriores, exceptuando la del gerente o director y responsable de contenido, que las ha desempeñado mi tutor en el proyecto.

3.4.2.- Recursos Hardware y Software:

Para el correcto funcionamiento de los robots, será necesario un intérprete de comandos de Perl, así como un sistema de gestión de base de datos cómo puede ser mySQL. La base de datos a tratar contendrá muchas entradas y será del orden de los 3Gbytes de tamaño.

La aplicación de consultas on-line correrá en un servidor ya existente y servirá para enriquecerlo con esta nueva utilidad. En dicho servidor se instalarán tanto los scripts de Perl que actualizan los datos, cómo la aplicación en sí.

Se deberá disponer de herramientas básicas para el almacenamiento y manejo de la base de datos, mySQL cómo comentamos anteriormente. Además será necesario que permita ejecutar rutinas en lenguaje Perl y PHP, ya que tanto los scripts de actualización de datos cómo las consultas a los mismos se realizarán empleando estos lenguajes.

Por supuesto deberá disponer de una buena conexión a internet ya que el tráfico de la aplicación será notable y se pretende en mayor medida que la respuesta de la misma sea rápida y efectiva.

El sistema operativo sobre el que correrá será Linux.

El acceso a la aplicación final se hará mediante web y será necesario que los clientes que accedan a la aplicación dispongan de la última versión de la máquina virtual Java ya que la interfaz se encuentra contenida en un Applet.

Deberán también disponer de una conexión a internet y de un explorador que les permita el acceso al servidor dónde se aloja la aplicación.

Cómo resumen, las necesidades tanto del cliente cómo del servidor serían las siguientes:

Servidor	**Cliente**
Espacio en disco del orden de los 3GB Sistema de Gestión de Bases de Datos (mySQL) Compiladores de Perl y PHP Óptima conexión a Internet Sistema operativo Linux Conexión FTP para acceso a ficheros remotos	Acceso a Internet Máquina Virtual de Java Explorador web

Cómo vemos, las necesidades pueden cubrirse sin ningún tipo de problemas, empleando en todo caso programas de libre distribución y código abierto.

3.5.- Interconexión y mantenimientos futuros:

Antes de interconectar la aplicación de consultas on-line e integrarla en el applet final, se construirá como applet independiente.
De esta forma, comprobaremos que su funcionalidad es correcta y que realiza los cruces y consultas de la forma deseada.

Este applet independiente recibirá unos genes de entrada de prueba que serán introducidos por teclado (indicando Símbolo de gen) y generará una salida en formato web con toda la información pertinente y necesaria para entender los genes resultados.

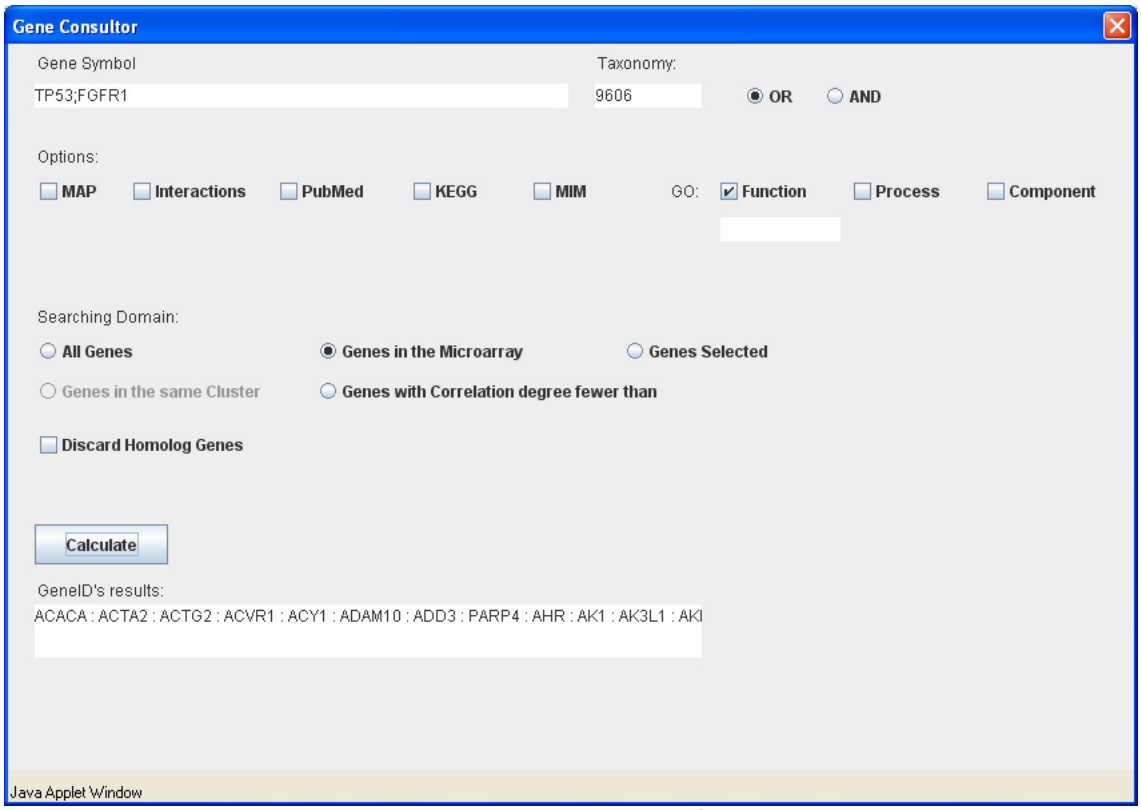


Imagen3.5.1: Prototipo de aplicación independiente

Para integrar esta aplicación en el applet final, deberemos cambiar las entradas y salidas de la misma:

- Las entradas serán los genes que estén clickados en el grafo de la microarray de la aplicación final, en lugar de una cadena de texto de los símbolos de gen.
- Las salidas deberán marcarse con diferente color en el grafo de la microarray, además de mostrar también la página web resumen con todos los resultados obtenidos.

Una vez enlazadas tanto las entradas como las salidas, la aplicación independiente formará parte del applet final, quedando ampliada su funcionalidad con esta herramienta.

El mantenimiento se basará principalmente en la ejecución rutinaria de los robots de actualización de la base de datos, controlando que la información almacenada en local sea coherente con la existente en las bases de datos remotas.

4.- ETAPAS DEL PROYECTO:

Basándonos en la planificación de tareas realizada anteriormente en el punto **3.3** *(Definición del marco temporal)*, veremos a continuación como enmarcarlas dentro de las etapas propias de la Ingeniería del Software.

En la dinámica de desarrollo del proyecto se mezclan estrategias de:

- Selección y organización de contenidos y documentación.
- La realización de contenido multimedia.
- La realización técnica y la programación.

En cualquier aplicación multimedia, a pesar de su integración, podemos diferenciar fácilmente estos tres componentes básicos, aunque se le presenten al usuario como un todo. La calidad del producto final en buena medida dependerá de la correcta integración de estos componentes.

A pesar de que nosotros enfocamos y estudiamos el proceso de una forma básicamente lineal, no hay que olvidar que el proceso de realización, sólo en parte es lineal, y que es el resultado de la interacción de todos los factores, que a menudo avanzan de forma paralela.

Cada una de las etapas a realizar en el desarrollo del proyecto estarán íntimamente relacionadas entre sí. Como mostramos en el punto **3.3.2** (*Diagrama de Gantt para el enfoque temporal)*, habrá tareas que se realizarán de forma simultánea y otras de forma lineal, existiendo una dependencia temporal entre ellas, de manera que si una tarea anterior no ha terminado, la posterior no podrá comenzar.

Etapas lineales de la producción:

Partiendo de un punto de vista temporal tenemos que imaginar la creación del producto dividida en 4 etapas o fases:

1. Planificación y coste:

Es la etapa conceptual en la cual se determinan muy bien los contenidos y los objetivos de la aplicación.

En esta fase, además de resolver los aspectos financieros, también hay que ponderar las alternativas tecnológicas y las estrategias de diseño y de desarrollo.
Antes de comenzar el proyecto, en esta etapa se debe planear qué habilidades de escritura, arte gráfico y otros objetos multimedia se requerirán *(punto 3 Estudio de Viabilidad).*

Propias de esta primera fase serán las siguientes tareas:

- Estudio del trabajo a realizar.
- Análisis de requisitos iniciales propuestos por el cliente.
- Investigación de las bases de datos del NCBI.
- Diseño lógico de la base de datos local.
- Diseño completo de la base de datos local, en base al estudio de la documentación de las bases de datos remotas del NCBI.

2. Diseño y producción:

Es la etapa en la que se define de forma exhaustiva el sistema o la aplicación. Durante el diseño se generan diversos documentos: guión temático, guión funcional, análisis técnico, guión media. En ésta etapa los aspectos relacionados con la aplicación y el diseño de la interacción son los más importantes.

Propias de esta primera fase serán las siguientes tareas:

- Implantación de un primer robot en un servidor de pruebas.
- Programación de los robots de actualización de datos locales a partir de Data Mining realizado sobre BD's del NCBI.
- Implantación final de los robots de actualización de datos en el servidor del departamento.
- Actualización de todos los datos remotos de genes en la base de datos local.
- Realización de consultas cruzando datos generados y datos de las BD del NCBI.
- Optimización de las consultas a las base de datos para obtener los resultados de los cruces.
- Realización de pantallas de la interfaz del applet.
- Disposición, integración y formato de pantallas finales en aplicación existente, mostrando resultados obtenidos.

3. Pruebas y Control de Calidad:

A medida que el desarrollo del proyecto o de la aplicación avanza deben comprobarse sistemáticamente todos los resultados parciales de tal modo que la realización no se podrá considerar concluida hasta que se hayan superado todas las pruebas previstas.

En esta etapa se genera un *feedback* o retroalimentación entre las tareas ya realizadas y las que están por realizar. De esta forma, puntos o apartados que habían quedado sin considerar en etapas anteriores serán añadidos o modificados.

Muy importante además es la detección de posibles errores de análisis o diseño previos, procediendo a su corrección si los hubiera.
Cuanto más atrás en el tiempo se encuentre el error, mayor será el coste asociado de recuperarlo ya que un mayor rango de tareas se habrán visto afectadas.

Es por tanto muy importante que se siga un control estricto en cada etapa del desarrollo del software para evitar posibles errores futuros.
Probando y controlando cada etapa en su debido momento, nos aseguramos de no encontrar futuras fallas en la aplicación.

4. Difusión:

La etapa final es la que se dedica a dar difusión. Hay que ocuparse, entre muchas otras cosas, de la publicidad y la distribución de la aplicación.
Habrá que determinar la forma de empaquetado más apropiado para distribuir la aplicación entre los usuarios finales.
Al tratarse este proyecto de un applet web, tan sólo será necesario alojarlo en el servidor del departamento, mencionado anteriormente en diversas ocasiones.

5.- ETAPA DE PLANIFICACIÓN Y COSTE:

La planificación de proyectos consiste en delimitar cada una de las fases de producción que, combinadas, darán forma a la idea original.
Cuantas más pequeñas sean estas tareas, más cómodo será el manejo y desarrollo de las mismas.

En el proceso de planificación de sistemas de información participarán los responsables de los procesos de la organización y los profesionales de los sistemas de información capaces de aportar las ventajas competitivas necesarias para la consecución del proyecto.

Antes de comenzar con la programación del proyecto multimedia, debemos medir su alcance y contenido.

Basándonos en el proceso de **Desarrollo de MÉTRICA Versión 3**, desarrollaremos todas las actividades y tareas que deben llevarse a cabo para construir un sistema, cubriendo desde el análisis de requisitos hasta la instalación del software.

MÉTRICA 3 trata dos tipos de desarrollos, estructurado y orientado a objetos.
Seguiremos el desarrollo orientado a objetos, teniendo en cuenta la mayoría de técnicas que contempla **UML 1.2.**

5.1.- MODELO DE ANÁLISIS (ANÁLISIS DE REQUISITOS):

El propósito de este proceso es conseguir la especificación detallada del sistema de información, a partir de una lista de requisitos y una serie de modelos que cubran todas las necesidades de los usuarios del sistema a desarrollar.

Estos requisitos serán la entrada para el proceso de **Diseño del Sistema de Información**.

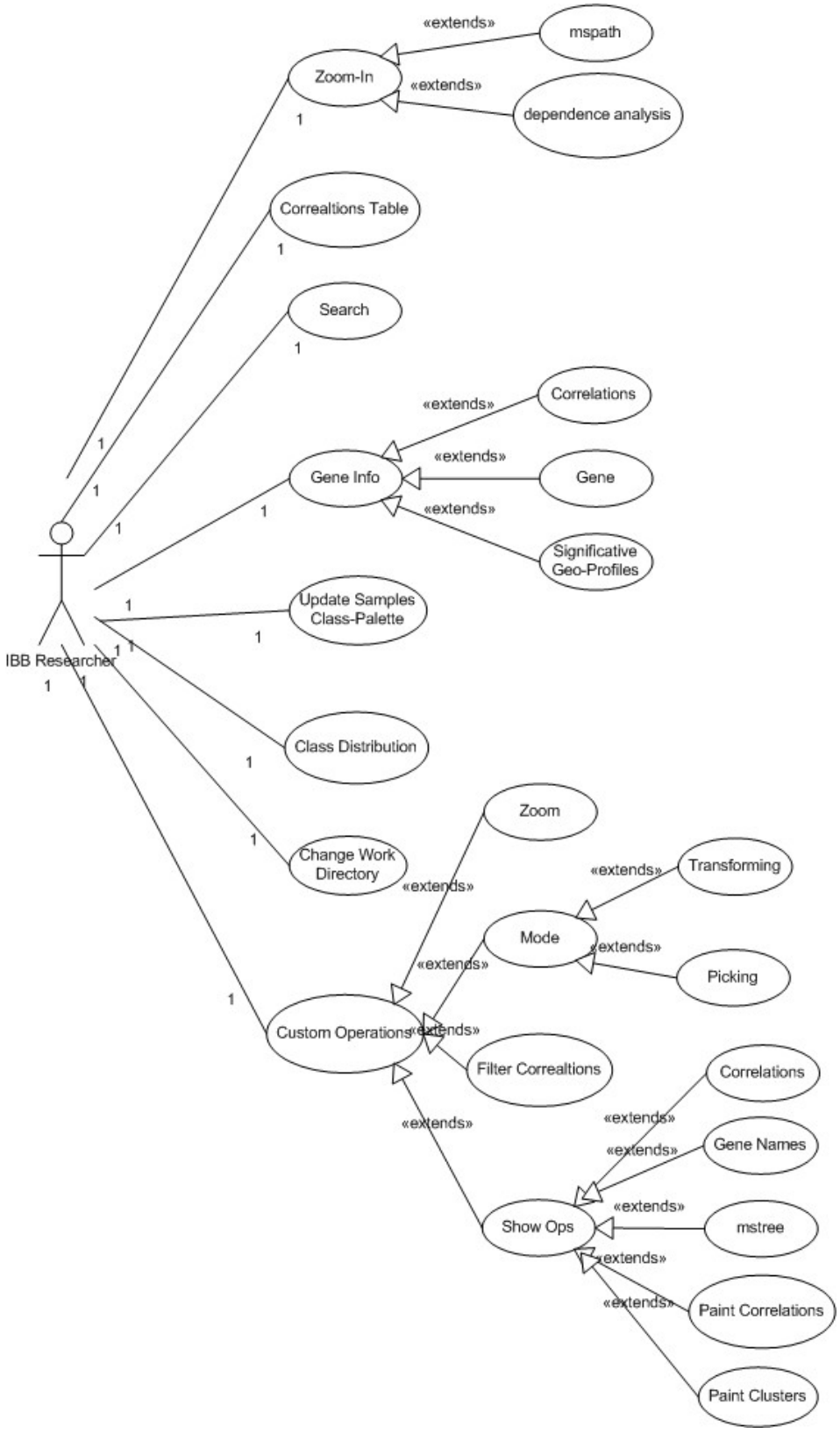
A continuación delimitaremos el alcance del sistema, generando para ello el catálogo de requisitos funcionales que el sistema de información debe cubrir.
Para facilitar el análisis del sistema, identificamos los subsistemas de análisis elaborando para ello los modelos de **Casos de Uso** y de **Clases**:

Los actores de nuestro sistema serán:

- IBB Researcher: Usuarios que accederán a la aplicación del departamento, principalmente investigadores que desean usar la herramienta de cruces de genes y de consultas on-line.
- Robots de actualización: Los robots serán aplicaciones programadas en Perl principalmente, encargados de la manutención y actualización de los datos locales y su concordancia con los mismos datos existentes en las bases de conocimiento remotas. Estos robots se ejecutarán en determinados momentos, previstos por el Cron del sistema operativo. Así pues, su activación en el sistema no será *aleatoria* como es el caso de los usuarios externos que desean usar la aplicación.

5.1.1.- Diagrama de Casos de Uso de la aplicación ya existente:

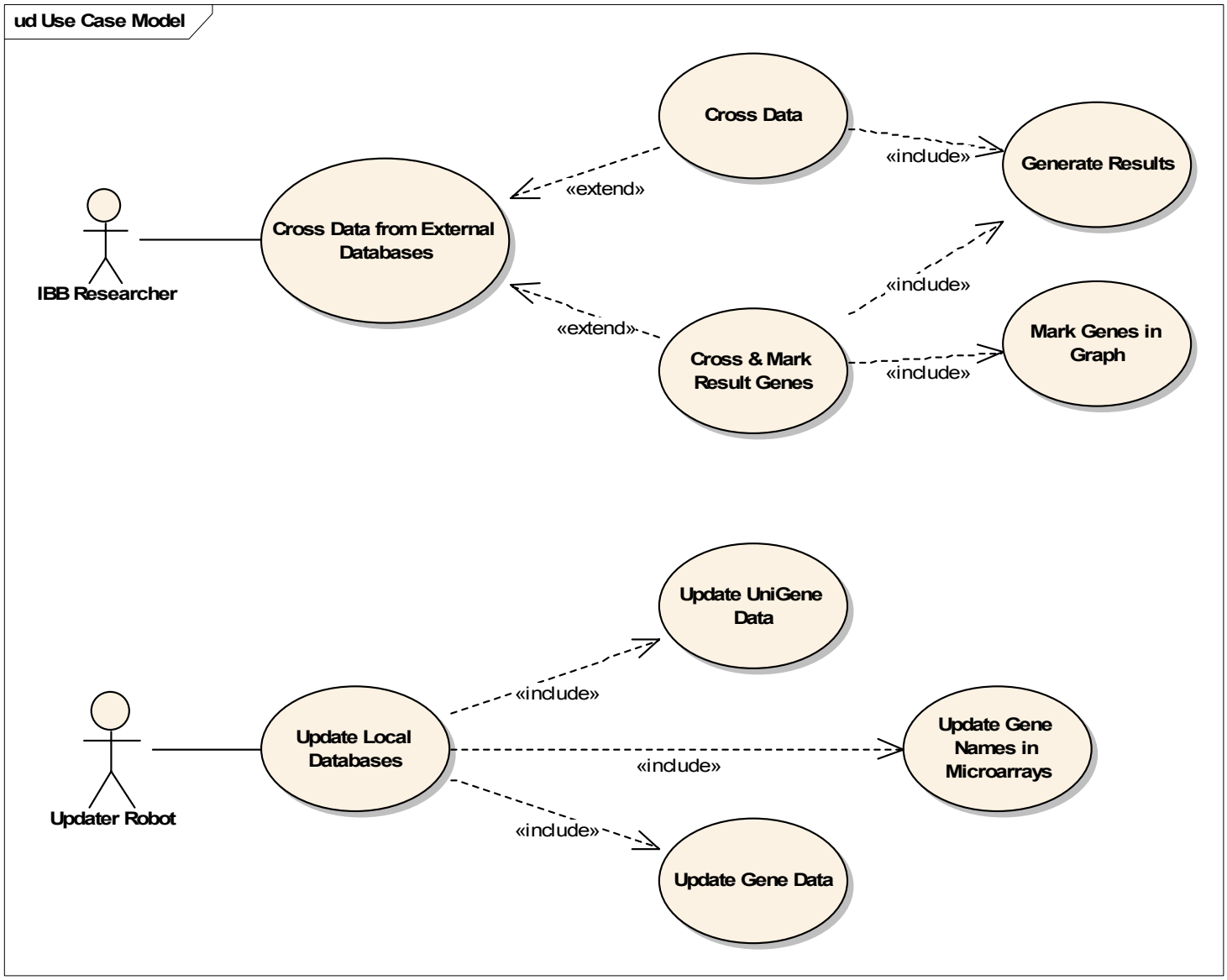
A continuación muestro los casos de uso ya implementados en un proyecto previo, que conforma la aplicación applet ya existente y a la cual agregaremos la nueva funcionalidad de cruces. No será objetivo de esta memoria definir en profundidad estos casos de uso ya que han sido documentados previamente.



5.1.2.- Diagrama de Casos de Uso:

Una vez tenemos claramente identificados los actores, pasamos a identificar los casos de uso con los que interactuarán.
Los casos de uso describen el comportamiento del sistema. En su interacción con el sistema, los actores inician los casos de uso, obteniendo un resultado de ellos.

Las actividades independientes que puede realizar nuestro sistema, correspondiéndose cada una de ellas con un caso de uso, son las siguientes:



Iniciados por el usuario investigador:

- Cruzar información de genes usando las Bases de datos biomédicas externas.
- Cruzar datos y generar resultados en formato web con estructura jerárquica.
- Cruzar datos, generar resultados en formato web y marcar genes resultados en grafo.

Iniciados por los robots de actualización:

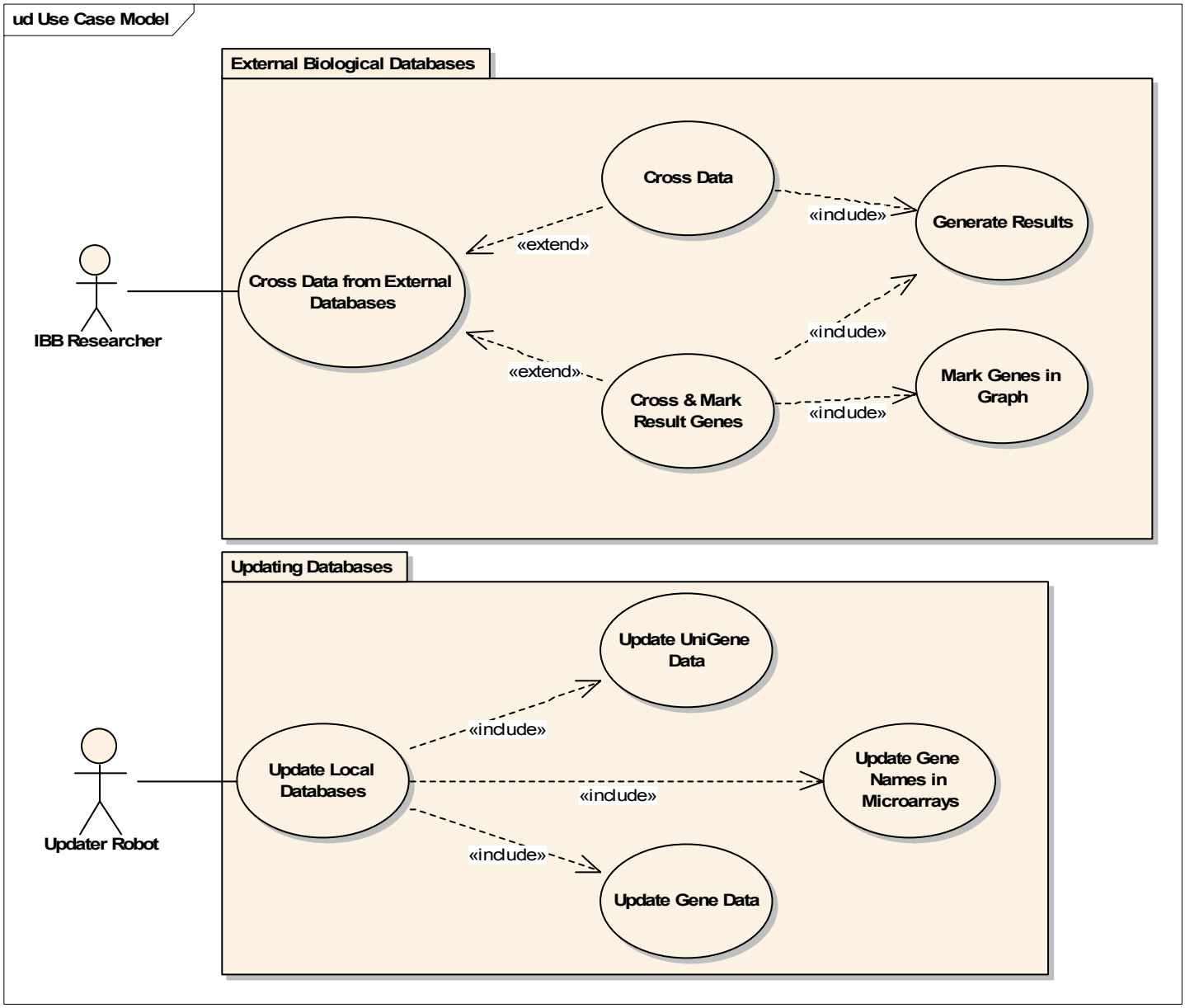
- Actualizar bases de datos locales:
    - Actualizar base de datos de Unigene.
    - Actualizar base de datos de Gene.
    - Actualizar nombres de genes en las microarrays.

5.1.3.- Identificación de los Subsistemas Funcionales:

Una vez tenemos descritos los casos de uso, pasamos a identificar los subsistemas funcionales en los que puede estructurarse el diagrama de casos de uso obtenido en el apartado anterior.

Para estructurar los diagramas nos basaremos en la funcionalidad de los casos de usos que lo conforman y en el actor que interactúa con ellos.

En el siguiente gráfico mostramos la agrupación de casos de uso:

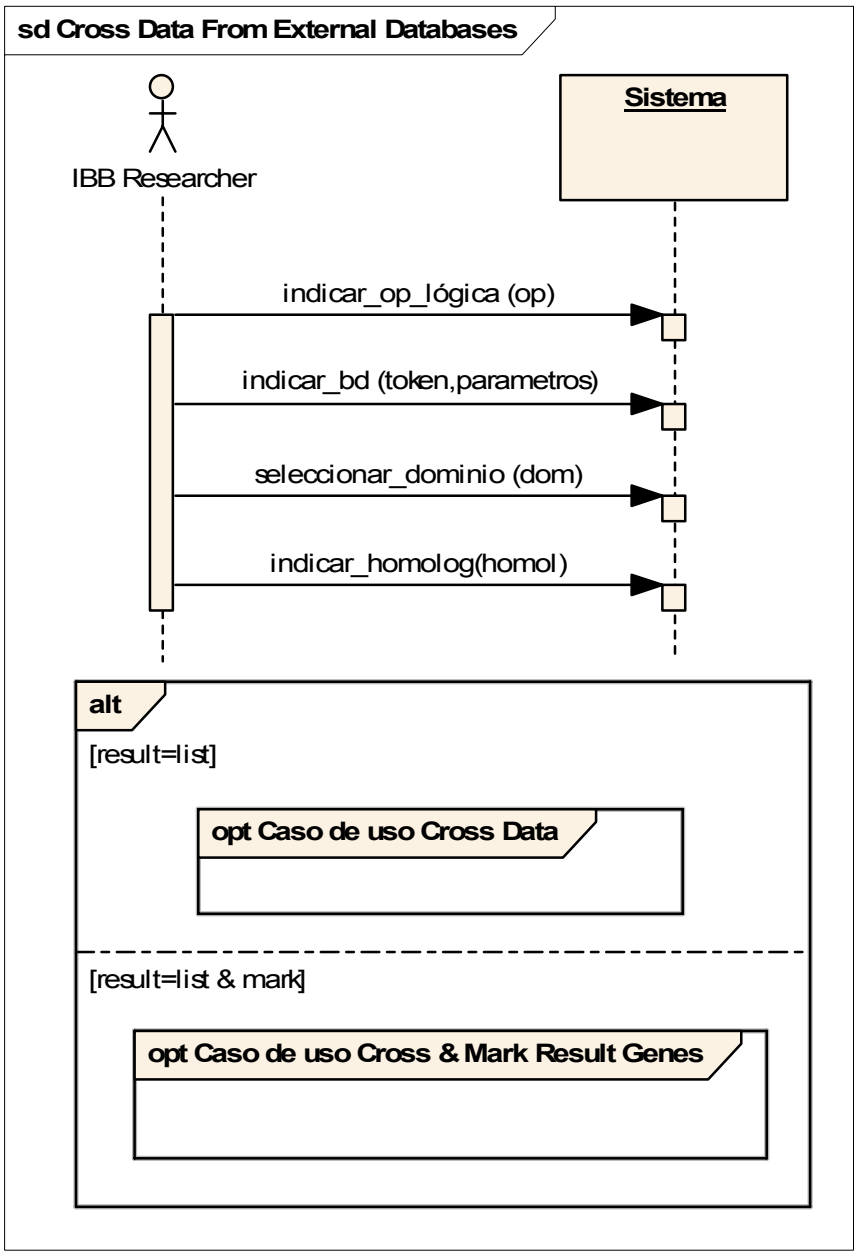


5.1.4.- Descripción detallada de cada Caso de Uso:

5.1.4.1.- *Cross Data From External Databases*:

Nombre del Caso	**Cross Data From External Databases**
Resumen	El usuario selecciona un conjunto de genes origen sobre los que realizar las consultas a bases de datos externas y obtener genes como resultado del cruce de información.
Dependencias	Se especializa en "Cross Data" y "Cross & Mark Result Genes".
Actores	IBB Researcher (Principal)
Precondiciones	El usuario debe acceder a la aplicación Bio Project vía web y haber marcado al menos un gen origen en el grafo 2D de la microarray.
Curso Normal	1. El usuario indica la operación lógica a realizar con los genes origen. 2. Establece las bases de datos que desea cruzar, indicando si cabe los parámetros de consulta para filtrar los resultados. 3. Selecciona el dominio dentro del cual obtener los resultados. 4. Indica si desea descartar genes homólogos de los resultados. 5. Selecciona cómo desea el resultado. a. Si sólo desea un listado de genes resultado, se llamará al caso de uso *Cross Data.* b. Si desea un listado de genes resultado y que aparezcan marcados en la aplicación, se llamará al caso de uso *Cross & Mark Result Genes.*
Postcondiciones	El usuario obtiene un conjunto de genes resultado.
Observaciones	El conjunto de genes resultado puede ser vacío (No se encontraron resultados).

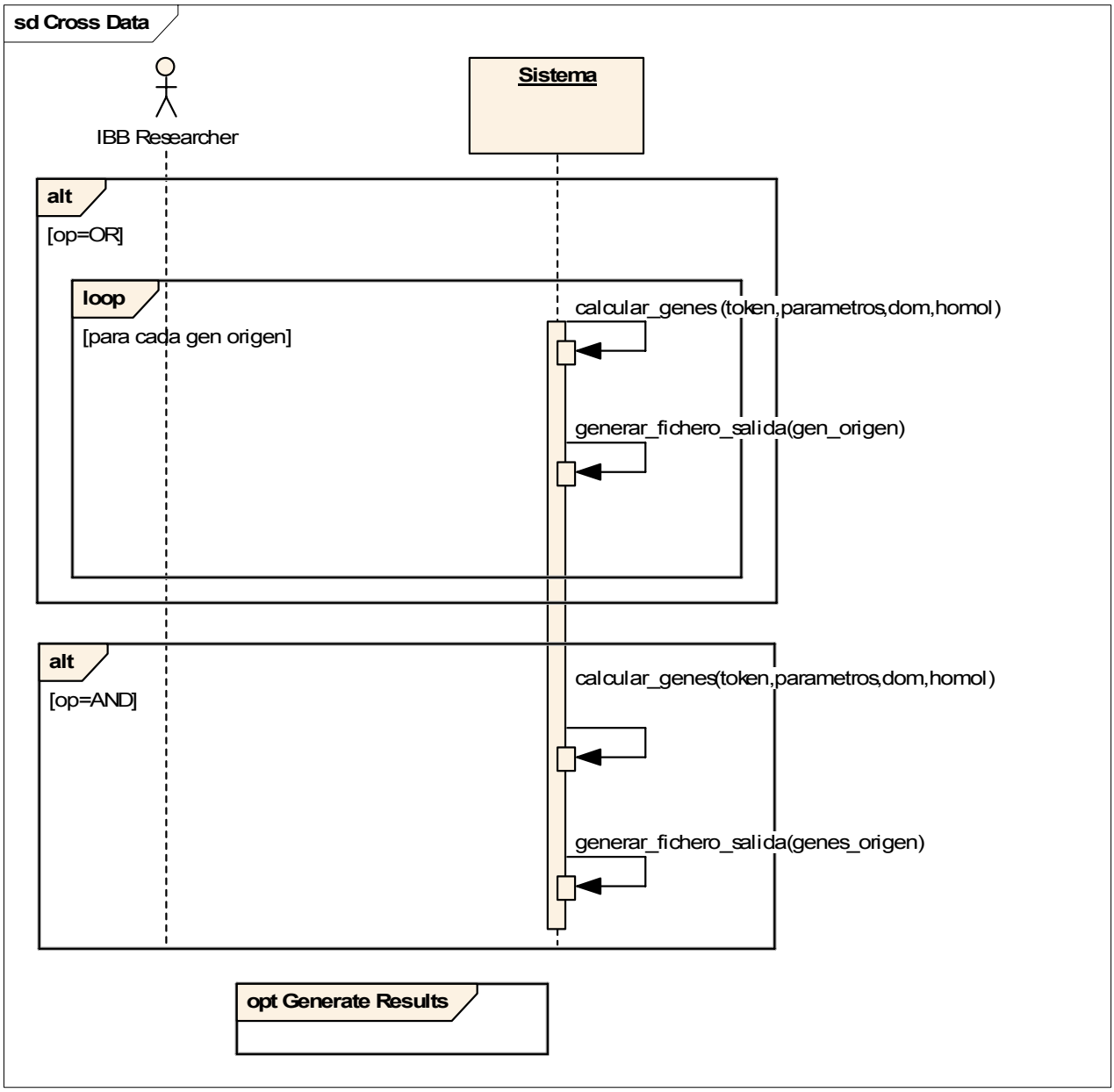
SISTEMA
indicar_op_lógica (op) – Indicar si hacer un Or o And con genes origen. **indicar_bd (token,parametros) –** Indicar conjunto de bd sobre las que hacer cruces y parámetros de búsqueda. **seleccionar_dominio (dom)** – Indicar el dominio de los resultados (Todos los genes, genes de la microarray, del mismo cluster, genes seleccionados, con mismo grado de correlación). **Indicar_homolog (homol) –** Indica si descartar genes homólogos del resultado. Según cómo quiera obtener los resultados el usuario, se llamará al caso de uso de obtener lista de resultados o al de obtener lista + marcar genes en el grafo de la aplicación.



5.1.4.2.- *Cross Data*:

Nombre del Caso	**Cross Data**
Resumen	A partir de los parámetros seleccionados por el usuario en el caso de uso que extiende, se obtendrán aquellos genes resultado que concuerdan con las condiciones de cruce indicadas.
Dependencias	Incluye caso de uso "Generate Results"
Actores	IBB Researcher (Principal)
Precondiciones	Deben haberse seleccionado los parámetros marcados en el caso de uso que extiende, *"Cross Data From External Databases".*
Curso Normal	1. Si se indicó la operación OR: a. Según los parámetros de cruce, se calculan los genes resultados para cada gen origen. b. Se genera un fichero de salida XML con los genes resultados y sus parámetros para cada gen origen. 2. Si se indicó la operación AND: a. Se calculan los genes resultados que cruzan con **todos** los genes origen. b. Se genera un único fichero de salida, con los genes resultados para el cruce con **todos** los genes origen. 3. Se llama al caso de uso *Generate Results.*
Postcondiciones	Se obtiene una lista de genes resultados.

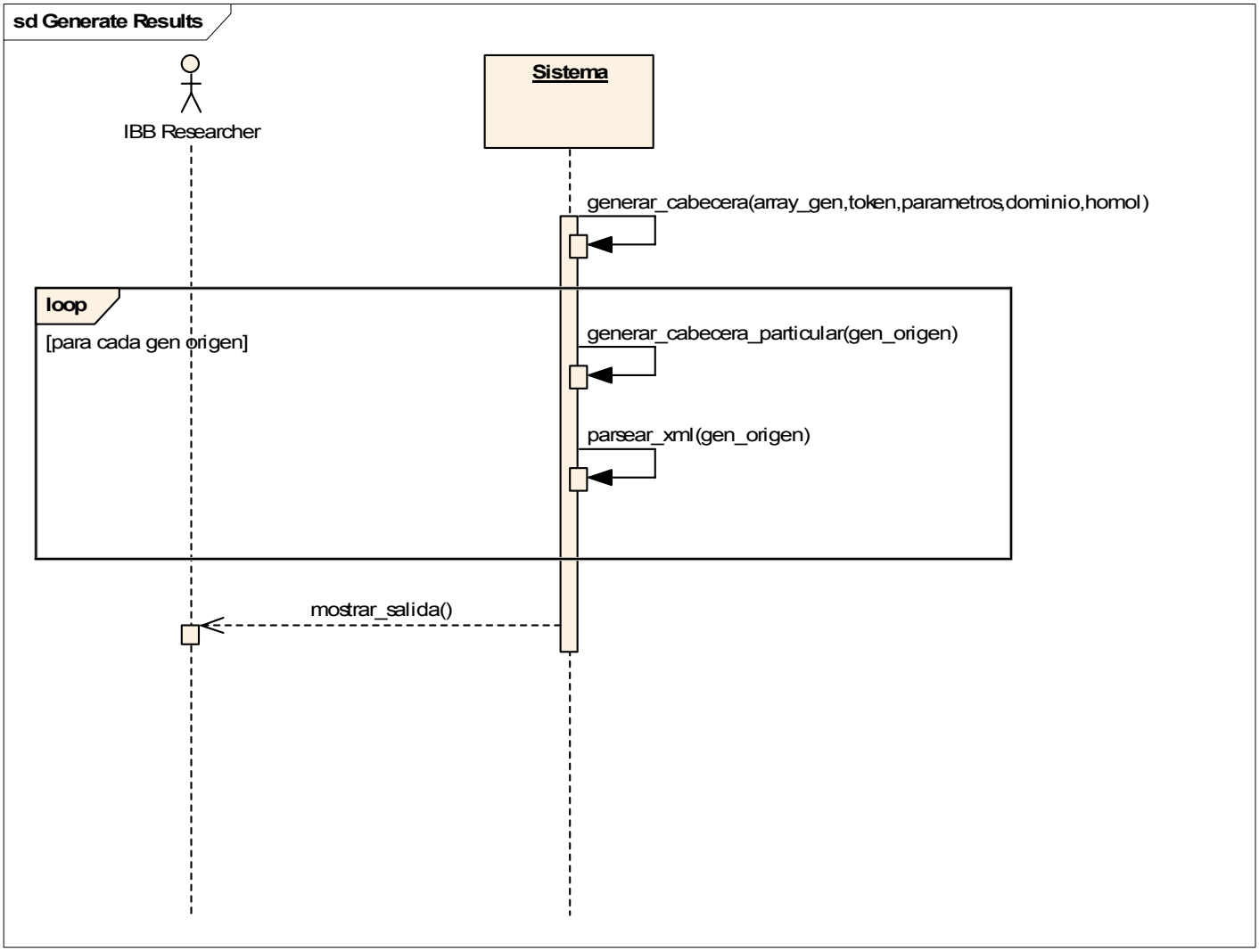
SISTEMA
calcular_genes(token,parametros,dom,homol) – Realiza los cruces de genes según los parámetros de la función, consultando en las bases de datos indicadas, con los parámetros asociados, buscando en el dominio indicado y descartando aquellos genes que son homólogos (si se elige dicha opción). **generar_fichero_salida (array_gen) –** Genera un fichero de salida XML con los genes resultados del cruce. Este archivo será el que posteriormente se lea para mostrar la salida en formato web manipulable. Tras calcular genes resultado y generar el fichero de salida, se llamará al caso de uso *Generate Results* que muestra los resultados en formato html.



5.1.4.3.- *Generate Results*:

Nombre del Caso	**Generate Results**
Resumen	Partiendo de los ficheros XML generados como salida y conteniendo los genes resultados, se creará una página html con toda la información pertinente para ser entendida. Contendrá una cabecera indicando los genes orígenes, las bases de datos seleccionadas, los parámetros indicados y el dominio de búsqueda.
Dependencias	Incluido en caso de uso "Cross Data" y "Cross & Mark Result Genes".
Actores	IBB Researcher (Principal)
Precondiciones	Se deben haber creado ficheros de salida que contienen los genes resultados para los genes orígenes marcados.
Curso Normal	1. Generar cabecera principal de información. 2. Para cada fichero de salida generado: a. Generar cabecera con información del gen origen para el que se muestran resultados. b. Parsear el fichero XML, generando enlaces, títulos de resultados obtenidos para cada base de datos seleccionada y nombres de genes solución.
Postcondiciones	Se obtiene una lista de genes resultados en formato html, con todos los links e información necesaria para su comprensión, dispuesta de forma jerárquica según las bases de datos seleccionadas.
Observaciones	Si los ficheros de salida están vacíos, no habrá genes resultados y la lista de salida estará vacía.

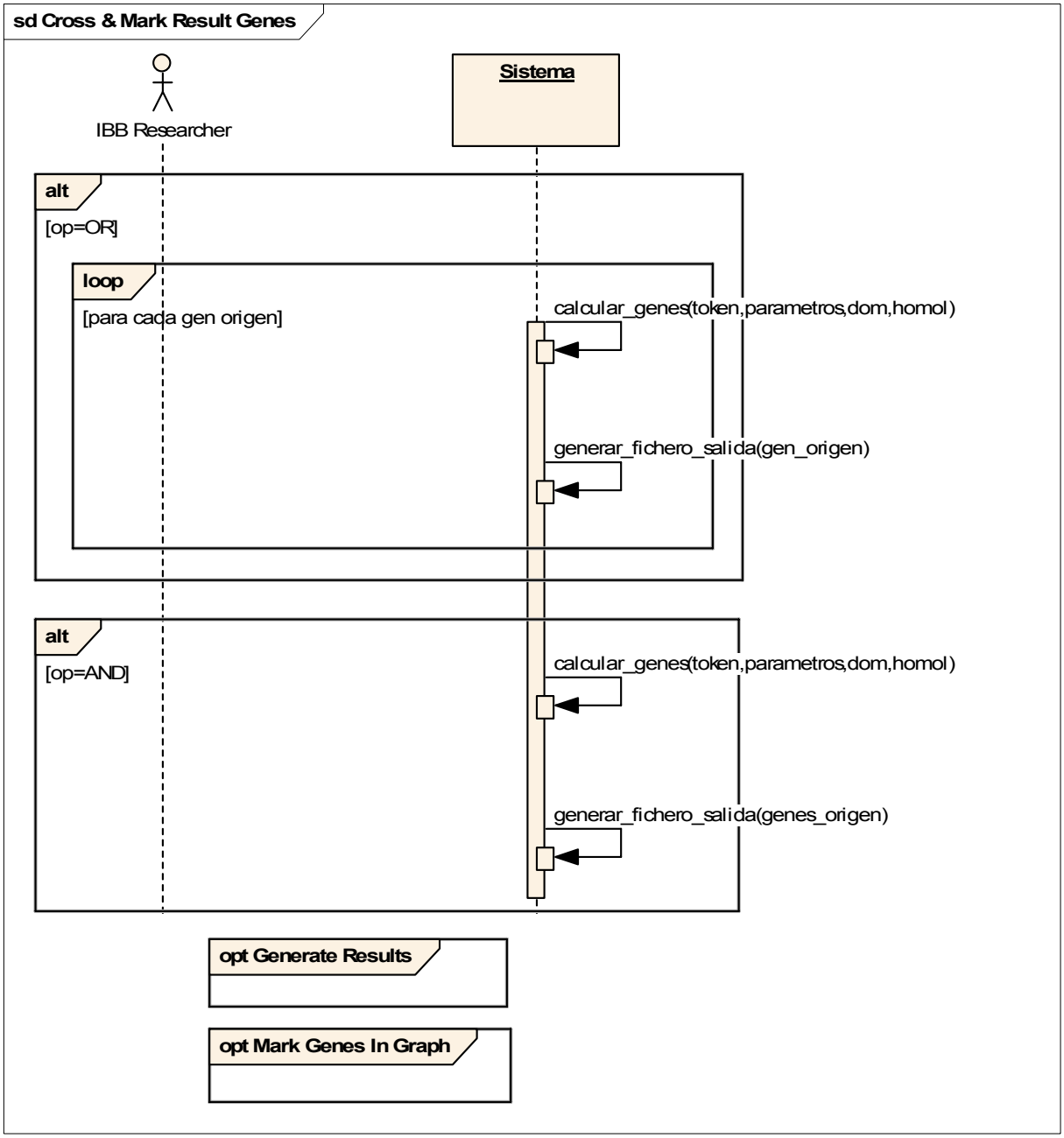
SISTEMA
generar_cabecera(array_gen,token,parametros,dom,homol) – Se genera en el informe de salida una cabecera principal, indicando los genes orígenes seleccionados, las bases de datos mediante las cuales vamos a cruzar los resultados, los parámetros particulares para dichas bases de datos, el dominio de búsqueda de los resultados y si se descartan los resultados homólogos o no. **generar_cabecera_particular(gen_origen) –** Para cada gen origen, se genera una cabecera indicando los resultados relacionados con dicho gen. Si se seleccionó la opción de AND para los genes orígenes, sólo habrá una cabecera con los resultados de las consultas para todos los genes orígenes. **parsear_xml(gen_origen) –** Los ficheros XML que contienen los resultados generados deberán parsearse para formatearlos en html, creando enlaces con información para genes, bases de datos, mapas genéticos, manipulación en el orden de la jerarquía a la hora de mostrar el resultado … **mostrar_salida() –** Muestra una página html con los genes resultado, ordenados jerárquicamente según las bases de datos por las que se han realizado los cruces. El usuario podrá cambian el orden de dicha jerarquía, organizando a su antojo cómo ver los resultados finales.



5.1.4.4.- *Cross & Mark Result Genes*:

Nombre del Caso	**Cross & Mark Result Genes**
Resumen	A partir de los parámetros seleccionados por el usuario en el caso de uso que extiende, se obtendrán aquellos genes resultado que concuerdan con las condiciones de cruce indicadas.
Dependencias	Incluye caso de uso "Generate Results" y "Mark Genes in Graph"
Actores	IBB Researcher (Principal)
Precondiciones	Deben haberse seleccionado los parámetros marcados en el caso de uso que extiende, *"Cross Data From External Databases".*
Curso Normal	1. Si se indicó la operación OR: a. Según los parámetros de cruce, se calculan los genes resultados para cada gen origen. b. Se genera un fichero de salida XML con los genes resultados y sus parámetros para cada gen origen. 2. Si se indicó la operación AND: a. Se calculan los genes resultados que cruzan con **todos** los genes origen. b. Se genera un único fichero de salida, con los genes resultados para el cruce con **todos** los genes origen. 3. Se llama al caso de uso *Generate Results.* 4. Se llama al caso de uso *Mark Genes in Graph.*
Postcondiciones	Se obtiene una lista de genes resultados y se marcan a su vez en el grafo de la microarray.

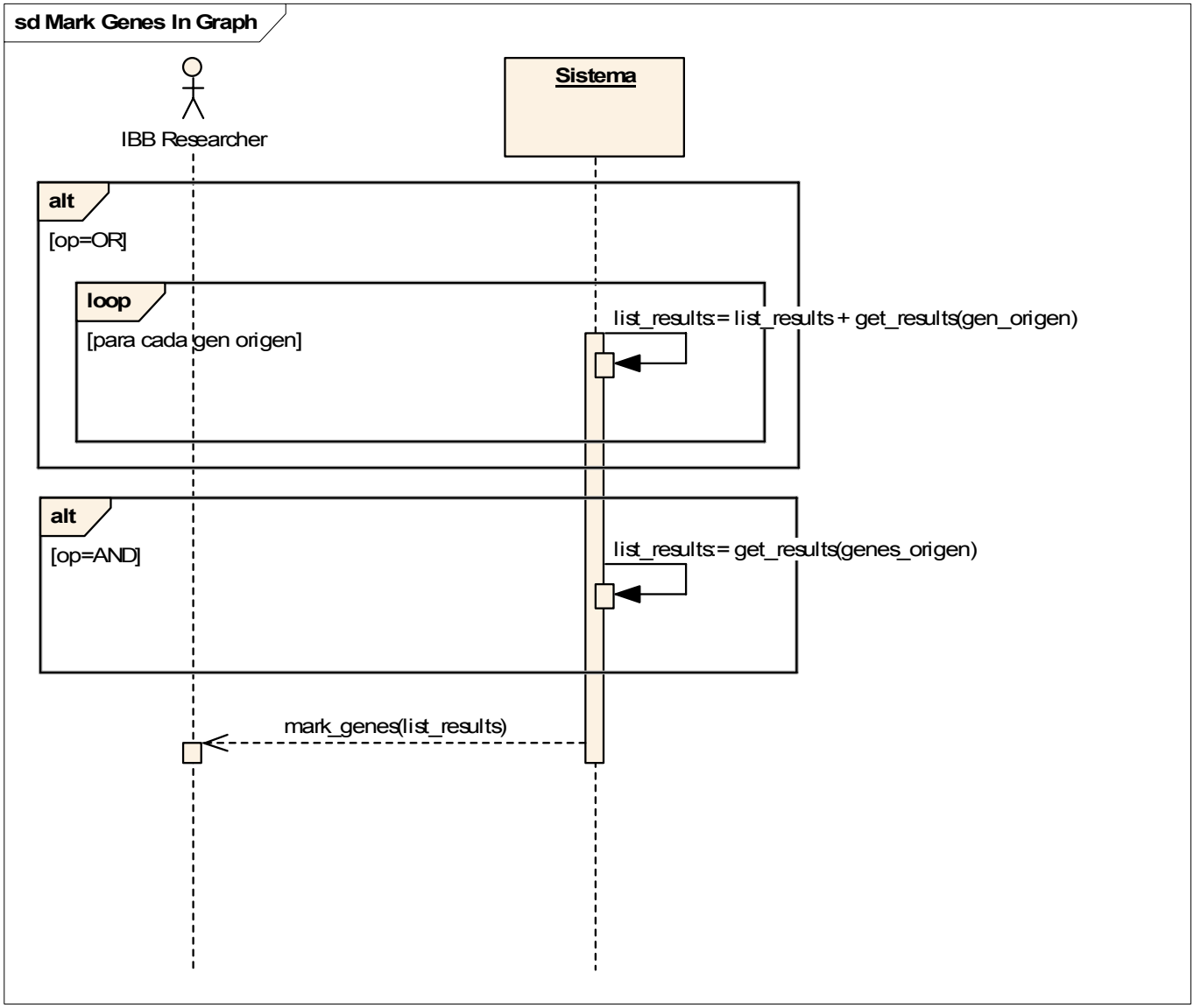
SISTEMA
calcular_genes(token,parametros,dom,homol) – Realiza los cruces de genes según los parámetros de la función, consultando en las bases de datos indicadas, con los parámetros asociados, buscando en el dominio indicado y descartando aquellos genes que son homólogos (si se elige dicha opción). **generar_fichero_salida (array_gen) –** Genera un fichero de salida XML con los genes resultados del cruce. Este archivo será el que posteriormente se lea para mostrar la salida en formato web manipulable. Tras calcular genes resultado y generar el fichero de salida, se llamará al caso de uso *Generate Results* que muestra los resultados en formato html. Además, se llamará al caso de uso *Mark Genes In Graph* para marcar los genes resultado en el grafo de la microarray.



5.1.4.5.- *Mark Genes In Graph*:

Nombre del Caso	**Mark Genes In Graph**
Resumen	Los genes obtenidos como resultado se marcan en el grafo de la microarray. Se hará distinción entre los genes marcados cómo orígenes y los marcados cómo resultado, mostrándolos con diferente color/forma.
Dependencias	Incluido en caso de uso "Cross & Mark Result Genes".
Actores	IBB Researcher (Principal)
Precondiciones	Se deben haber creado ficheros de salida que contienen los genes resultados para los genes orígenes marcados.
Curso Normal	1. Si se indicó la operación OR: a. Para cada gen origen, obtener de su fichero XML correspondiente los GeneID's de los genes resultado que se han generado. 2. Si se indicó la operación AND: a. Obtener del único fichero de salida, los GeneID's de los genes resultados obtenidos como cruce de todos los genes origen. 3. Marcar los genes resultados en el grafo de la microarray.
Postcondiciones	La lista de genes resultados se marcan en el grafo de la microarray, distinguiendo entre genes marcados como origen y genes obtenidos como resultado.

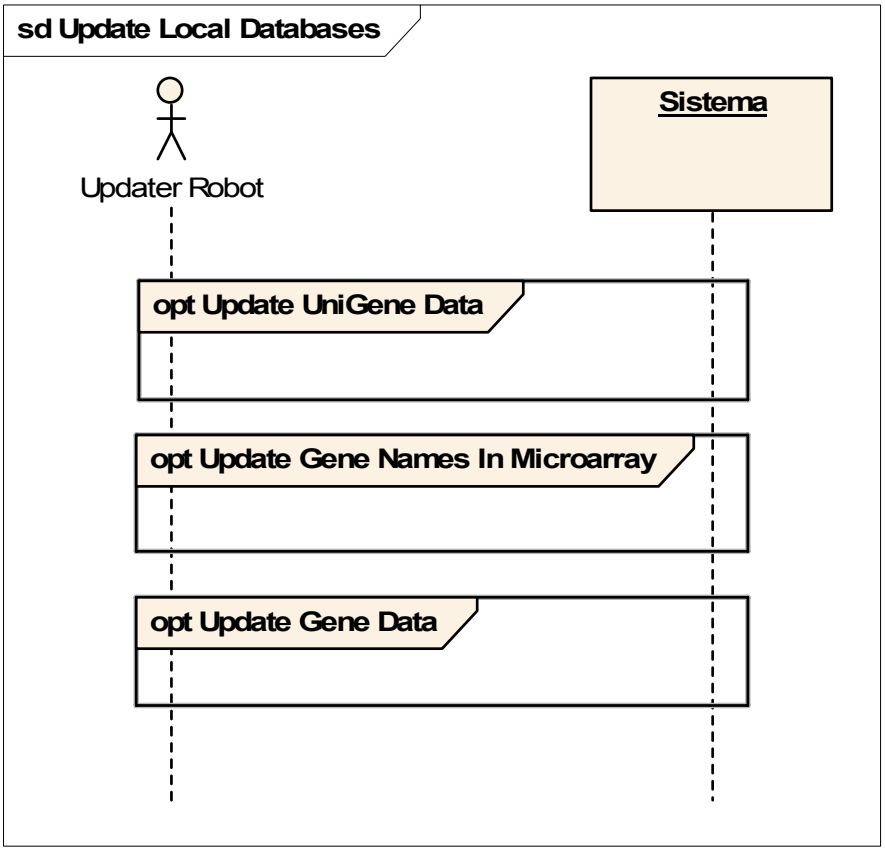
SISTEMA
get_results(gen_origen) – Dado un gen origen, obtiene de su fichero XML correspondiente generado anteriormente, una lista de GeneID's resultado del cruce de información. **mark_genes(list_results) –** A partir de una lista de GeneID's resultados, marca dichos genes en el grafo de la microarray. Los genes orígenes de la operación se marcarán de forma diferente que los genes resultados, de manera que puedan distinguirse bien a ojos del usuario.



5.1.4.6.- *Update Local Databases*:

Nombre del Caso	**Update Local Databases**
Resumen	Se lanza el robot de actualización de bases de datos locales, que se conectará mediante FTP al servidor del NCBI, para bajarse aquellos ficheros con información relevante para nuestra bases de datos local. Actualizará los datos almacenados de Unigene y Gene, y además actualizará los nombres de los genes de microarrays que existan.
Dependencias	Incluye los casos de uso *"Update Unigene Data", "Update Gene Data" y "Update Gene Names In Microarrays"*.
Actores	Updater Robot (Principal)
Precondiciones	El Cron del sistema operativo tiene una entrada para ejecutar el robot de actualización una vez al mes.
Curso Normal	Lanza en orden consecutivo los 3 módulos de actualización: 1. Update UniGene Data 2. Update Gene Names In Microarrays 3. Update Gene Data
Postcondiciones	Las bases de datos locales quedan actualizadas.
Observaciones	El estado de la base de datos local no quedará correcto hasta que hayan terminado los tres módulos de actualización, pudiéndose usar la aplicación en dicho periodo, aunque los resultados puede que sean correctos *temporalmente*.

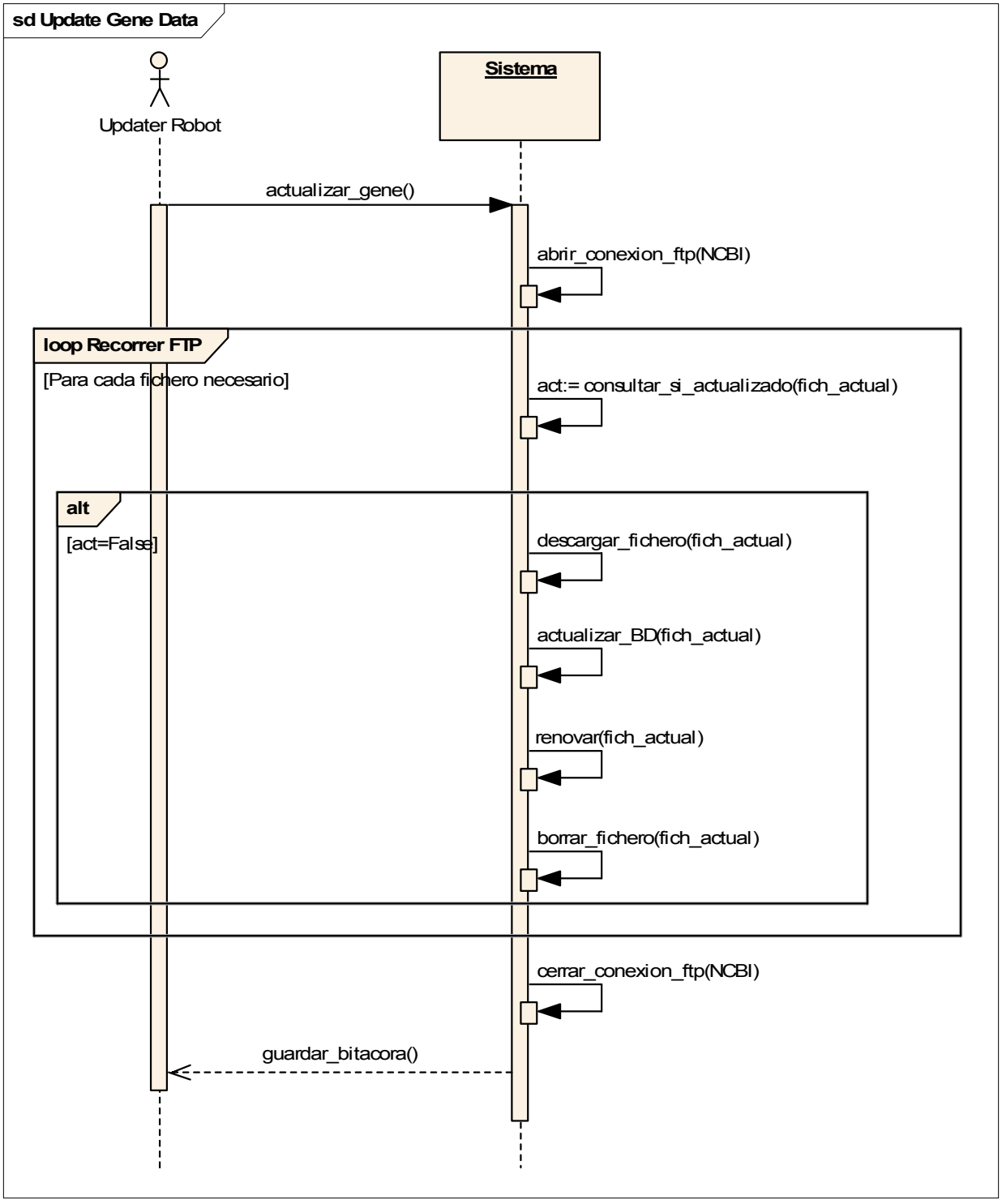
SISTEMA
Al lanzarse el robot, se llama secuencialmente a cada uno de los casos de uso: 1. Update UniGene Data: Actualiza datos referentes a UniGene (números de secuencia asociados a GeneID's). 2. Update Gene Names In Microarrays: actualiza los nombres de genes de los ficheros de microarray que existen en la aplicación. 3. Update Gene Data: Actualiza los datos locales referentes a Gene (pubmed,omim,interactions,map,amiGo,Kegg).



5.1.4.7.- *Update Gene Data*:

Nombre del Caso	**Update Gene Data**
Resumen	Se actualizan las bases de datos locales de Gene, toda la información actualizada correspondiente a cada GeneID. Para ello, crearemos una conexión FTP con el sitio gene del NCBI. Se bajarán los ficheros no actualizados, que contienen información necesaria para realizar las consultas cruzadas sobre los genes. A partir de estos ficheros, se hará un parsing de la información relevante que contengan y se actualizará en local.
Dependencias	Incluido en el caso de uso *"Update Local Databases"*.
Actores	Updater Robot (Principal)
Precondiciones	La estructura lógica y física de la base de datos se encuentra creada en el Sistema de gestión de Bases de datos y se puede acceder a ella.
Curso Normal	1. Abrir conexión FTP con el NCBI. 2. Para cada fichero necesario del FTP: a. Comprobar si se encuentra actualizado en la BD local. b. Si no es así, descargarlo. c. Obtener del fichero la información necesaria y actualizarla en la BD. d. Borrar fichero descargado y renovar la fecha de descarga en la Base de datos. 3. Cerrar conexión FTP.
Postcondiciones	Las bases de datos locales con información de Gene quedan actualizadas.
Observaciones	Los ficheros a bajarse del FTP están preestablecidos, tras haberse realizado un estudio de cuales se puede obtener información relevante para el desarrollo de la aplicación de cruce de información *(Ver apartado Diseño)*.

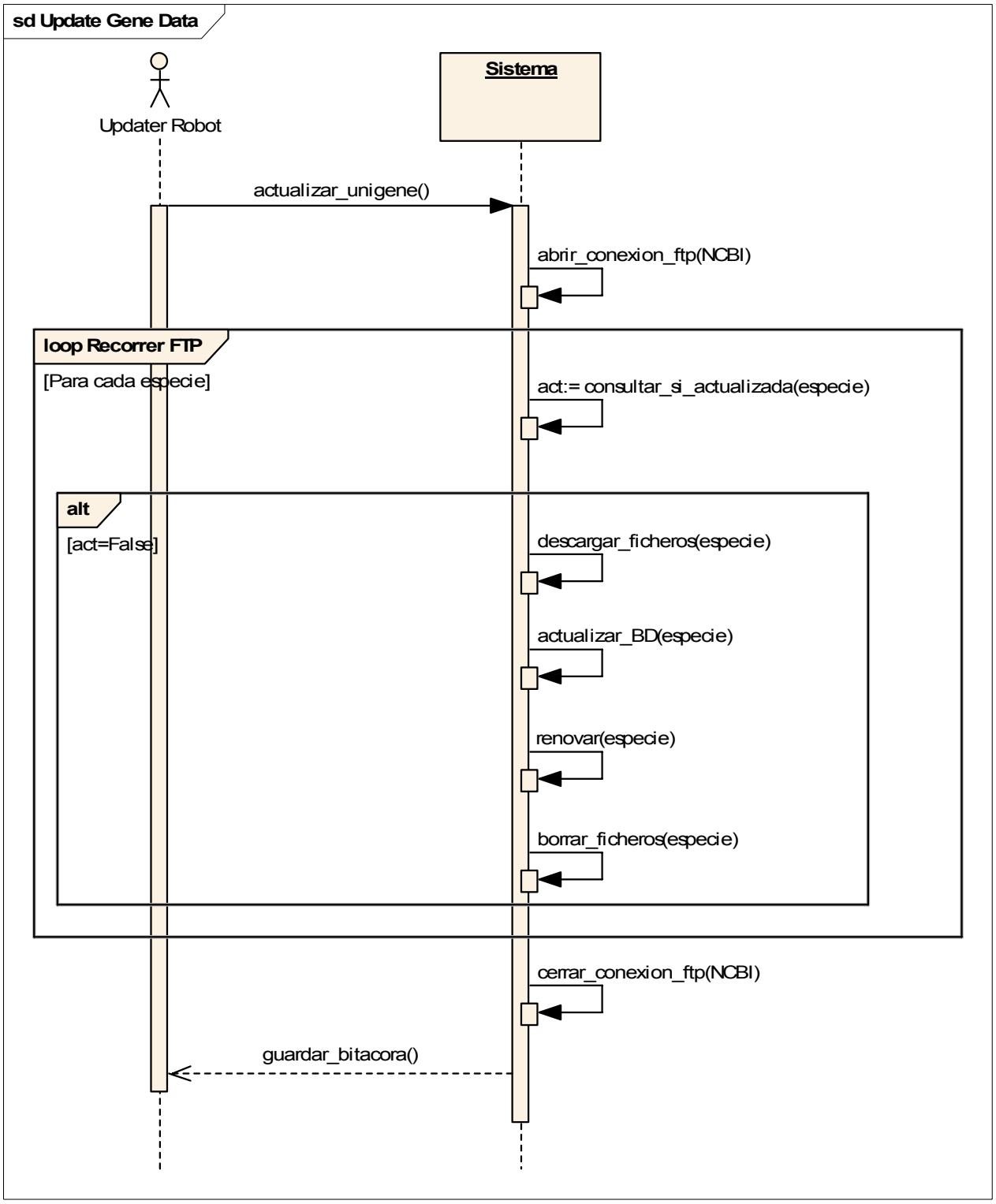
SISTEMA
actualizar_gene() – Da la orden para comenzar a actualizar los datos locales. **abrir_conexion_ftp(NCBI) –** Abre una conexión FTP con el servidor del NCBI. **consultar_si_actualizado(fich_local) –** Comprueba si el fichero local a descargarse ha sido actualizado desde la última actualización local, para no volver a descargarlo si es el caso. **descargar_fichero(fich_actual) –** Descargar el fichero del FTP a un directorio temporal en la cuenta del usuario *Robot*. **actualizar_BD(fich_actual) –** Una vez descargado el fichero que contiene la información, se tratará su contenido, utilizando la información relevante que contenga para actualizar la base de datos local. **renovar(fich_actual) –** Modificar la fecha de descarga del fichero, de manera que llevemos un control de fechas de cada fichero descargado, para no volver a descargarlo en el caso de que no haya sufrido ninguna modificación desde la última vez. **borrar_fichero(fich_actual) –** Eliminar fichero del directorio temporal dónde se ha descargado. No es necesario mantenerlo ya que la información relevante que contenía ya ha sido actualizada en la Base de datos y además, hemos guardado la fecha de descarga. Nos ahorramos así mucho espacio en disco ya que el tamaño de los ficheros a descargar puede llegar a ser del tamaño de los GigaBytes. **cerrar_conexion_ftp(NCBI) –** Cerrar la conexión FTP con el servidor del NCBI. **guardar_bitácora() –** Se generará un fichero de bitácora que servirá como salida para el Robot. En este fichero se almacenarán las tareas realizadas (fecha y hora de descarga de ficheros, actualización de tablas dentro de la base de datos, errores encontrados…).


sd Update Gene Data
Updater Robot
Sistema
actualizar_gene()
abrir_conexion_ftp(NCBI)
loop Recorrer FTP
[Para cada fichero necesario]
act:= consultar_si_actualizado(fich_actual)
alt
[act=False]
descargar_fichero(fich_actual)
actualizar_BD(fich_actual)
renovar(fich_actual)
borrar_fichero(fich_actual)
cerrar_conexion_ftp(NCBI)
guardar_bitacora()

5.1.4.8.- *Update UniGene Data*:

Nombre del Caso	**Update UniGene Data**
Resumen	Se actualizan las bases de datos locales de UniGene, toda la información actualizada correspondiente a cada especie (Números de secuencia asociados a Clusters de Unigene y a GeneID's). Para ello, crearemos una conexión FTP con el sitio UniGene del NCBI. Se bajarán los ficheros no actualizados, que contienen información necesaria para realizar las consultas cruzadas sobre los genes. A partir de estos ficheros, se hará un parsing de la información relevante que contengan y se actualizará en local.
Dependencias	Incluido en el caso de uso *"Update Local Databases"*.
Actores	Updater Robot (Principal)
Precondiciones	La estructura lógica y física de la base de datos se encuentra creada en el Sistema de gestión de Bases de datos y se puede acceder a ella.
Curso Normal	1. Abrir conexión FTP con el NCBI. 2. Para cada especie del FTP: a. Comprobar si los ficheros de especie se encuentran actualizados en la BD local. b. Si no es así, descargarlos. c. Obtener de los ficheros de la especie la información necesaria y actualizarla en la BD. d. Borrar ficheros descargados y renovar la fecha de descarga en la Base de datos. 3. Cerrar conexión FTP.
Postcondiciones	Las bases de datos locales con información de UniGene quedan actualizadas.
Observaciones	Los ficheros a bajarse del FTP están preestablecidos, tras haberse realizado un estudio de cuales se puede obtener información relevante para el desarrollo de la aplicación de cruce de información *(Ver apartado Diseño)*.

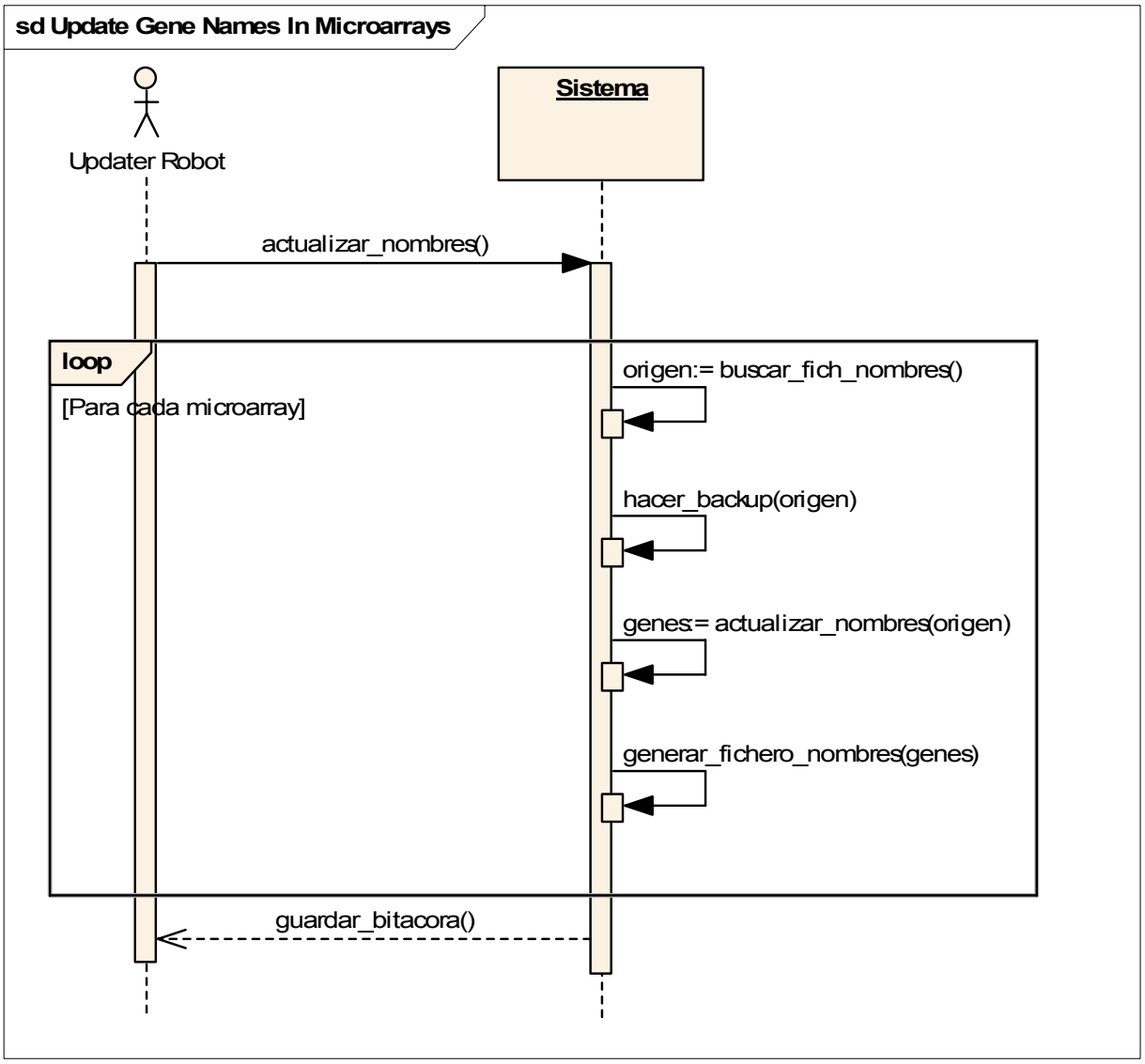
SISTEMA
actualizar_unigene() – Da la orden para comenzar a actualizar los datos locales. **abrir_conexion_ftp(NCBI) –** Abre una conexión FTP con el servidor del NCBI. **consultar_si_actualizada(especie) –** Comprueba si los ficheros de la especie a descargarse han sido actualizados desde la última actualización local, para no volver a descargarlos si es el caso. **descargar_ficheros(especie) –** Descargar los ficheros con información UniGene de la especie del FTP a un directorio temporal en la cuenta del usuario *Robot*. **actualizar_BD(especie) –** Una vez descargados los ficheros que contienen la información, se tratará su contenido, utilizando la información relevante que contengan para actualizar la base de datos local. **renovar(especie) –** Modificar la fecha de descarga de los ficheros, de manera que llevemos un control de fechas de cada fichero descargado, para no volver a descargarlo en el caso de que no haya sufrido ninguna modificación desde la última vez. **borrar_ficheros(especie) –** Eliminar ficheros de la especie del directorio temporal dónde se han descargado. No es necesario mantenerlos ya que la información relevante que contenían ya ha sido actualizada en la Base de datos y además, hemos guardado la fecha de descarga. Nos ahorramos así mucho espacio en disco ya que el tamaño de los ficheros a descargar puede llegar a ser del tamaño de los GigaBytes. **cerrar_conexion_ftp(NCBI) –** Cerrar la conexión FTP con el servidor del NCBI. **guardar_bitácora() –** Se generará un fichero de bitácora que servirá como salida para el Robot. En este fichero se almacenarán las tareas realizadas (fecha y hora de descarga de ficheros, actualización de tablas dentro de la base de datos, errores encontrados…).


sd Update Gene Data
Updater Robot
Sistema
actualizar_unigene()
abrir_conexion_ftp(NCBI)
loop Recorrer FTP
[Para cada especie]
act:= consultar_si_actualizada(especie)
alt
[act=False]
descargar_ficheros(especie)
actualizar_BD(especie)
renovar(especie)
borrar_ficheros(especie)
cerrar_conexion_ftp(NCBI)
guardar_bitacora()

5.1.4.9.- *Update Gene Names In Microarrays*:

Nombre del Caso	**Update Gene Names In Microarrays**
Resumen	Se recorrerán los directorios locales de microarrays, buscando en ellos los ficheros .genesorig (contienen nombre de los genes de dicha microarray), sustituyendo cada línea por el nombre de gen actualizado correspondiente al gen de dicha línea (formato de nombre de gen: "*Symbol: Nombre Del Gen"*).
Dependencias	Incluido en el caso de uso *"Update Local Databases"*.
Actores	Updater Robot (Principal)
Precondiciones	La estructura lógica y física de la base de datos se encuentra creada en el Sistema de gestión de Bases de datos y se puede acceder a ella. Debe existir información correspondiente en la Base de datos local de UniGene, ya que será de ésta de la que se obtendrán los nombres de genes asociados a un GeneID.
Curso Normal	1. Para cada directorio de microarray: a. Obtener fichero .genesorig. b. Para cada línea del fichero, obtener el nombre de gen actualizado. c. Guardar en fichero .genes los nuevos nombres de gen.
Postcondiciones	Los ficheros .genesorig de las microarrays existentes quedan actualizados correctamente con los nombres de gen en el fichero .genes. El fichero .genesorig queda tal cual, sin modificar.
Observaciones	Existe un directorio en el servidor del IBB en el que se almacena información acerca de las microarrays existentes. En estos directorios existe un fichero de nombres para cada microarray (.genesorig). Este fichero es el que se usa para generar el fichero de nombres de genes (.genes). Cada línea de este fichero corresponde a un gen dentro de la microarray, el nombre de dicho gen actualizado. *(Ver apartado de Diseño).* El fichero .genesorig debe permanecer inalterado, ya que es a partir de los códigos de gen que contiene en cada línea cómo podremos obtener los nuevos nombres de gen en las sucesivas ejecuciones de este caso de uso.

SISTEMA
actualizar_nombes() – Da la orden para comenzar a actualizar los ficheros de nombres de las microarrays existentes en el directorio local correspondiente. **buscar_fichero_nombres() –** Obtener fichero .genesorig de la microarray actual, a partir del cual obtener los nombres para cada gen (cada línea del fichero). **hacer_backup(origen) –** Realizar una copia de seguridad del fichero .genesorig ya que este fichero debe permanecer inalterado. Esto es así para que, en sucesivas actualizaciones de nombres, siempre podamos partir de este fichero y, a partir de la información que contiene, generar los nuevos nombres que hayan sido actualizados en la base de datos. **actualizar_nombres(origen) –** Para cada línea del fichero .genesorig (que contendrá una serie de códigos de gen), consultaremos en la base de datos local que contiene información sobre UniGene. A partir de los códigos de secuencia o de especie que aparecen en cada línea, obtendremos de la base de datos su nombre y símbolo de gen asociado. **generar_fichero_nombres(genes) –** Con los nuevos nombres de gen encontrados, generamos en el directorio de la microarray un fichero .genes que sólo contendrá nombres de gen, uno por cada línea. Este fichero no contendrá códigos de secuencia o especie, ya que sólo el fichero .genesorig los conservará. A partir de estos códigos, en sucesivas actualizaciones de nombres, obtendremos futuros nombres que hayan sido actualizados en la base de datos. **guardar_bitácora() –** Se generará un fichero de bitácora que servirá como salida para el Robot. En este fichero se almacenarán las tareas realizadas (fecha y hora de descarga de ficheros, actualización de tablas dentro de la base de datos, errores encontrados…).


sd Update Gene Names In Microarrays
Sistema
Updater Robot
actualizar_nombres()
loop
[Para cada microarray]
origen:= buscar_fich_nombres()
hacer_backup(origen)
genes:= actualizar_nombres(origen)
generar_fichero_nombres(genes)
guardar_bitacora()

5.2.- MODELO CONCEPTUAL DE DOMINIO:

Será en esta sección dónde desarrollemos un estudio pormenorizado de las diferentes bases de datos que tomarán partido en nuestra aplicación.

Analizaremos en profundidad las bases de datos externas que necesitaremos, los campos con información útil, sus estructuras lógicas y físicas.
Una vez que conozcamos toda la información necesaria para poder realizar la aplicación de cruces on-line, estableceremos el modelo lógico y físico de nuestra base de datos local.

Los robots de actualización serán los encargados, cada cierto tiempo, de acceder a la información que aquí estudiemos, descargarla y actualizarla en las bases de datos locales.

La funcionalidad de los cruces on-line será obtener como genes resultado aquellos que cumplan una serie de condiciones respecto a unos genes origen. Estas condiciones serán las que harán de *filtro* entre todos los genes existentes, quedándonos con aquellos que *superen* dicho filtro.

Nuestras bases de datos de trabajo serán la Gene, UniGene, PubMed, OMIM, Homologene y KEGG. Todas ellas son propiedad del NCBI, excepto KEGG que será accesible a través de su propia página web.

La información que nos interesará principalmente está contenida en la base de datos de **Gene**. De aquí necesitaremos conocer:

- Información básica del gen: GeneID, Símbolo, Nombre, Cromosoma en el que se encuentra y Taxonomía.
- Interacciones: genes con los que interacciona un gen.
- Gene Ontology: función, proceso y componente de un gen.
- Genes que están en misma función, proceso y/o componente que un gen.
- Aproximate Location: obtener genes vecinos a un gen dado.

De la base de datos **UniGene** necesitaremos:

- De cada especie, es necesario hacer la correspondencia de diversos tipos de código a GeneID.
- Necesitaremos las tablas en las que obtener un código GeneID a partir de diferentes códigos de secuencia de gen que se pueden encontrar en la microarray.
- El principal objetivo de esta base de datos es unificar los códigos y emplear siempre el GeneID para referirnos a un gen.

De **PubMed** será necesario:

- Publicaciones en las que aparece un gen dado.
- Genes que aparecen en las mismas publicaciones que un gen dado.
- Título de la publicación.
- Dirección web del NCBI para las publicaciones.

OMIM se usará para cruzar genes por patologías:

- Patologías de un gen dado.
- Genes que aparecen en misma patología que un gen dado.
- Nombre y descripción de las patologías.
- Dirección web del NCBI para las patologías.

La base de datos **Homologene** se empleará como criba final, para descartar genes que sean homólogos entre sí entre los resultados obtenidos. Será necesario pues conocer:

- Genes homólogos a un gen dado.

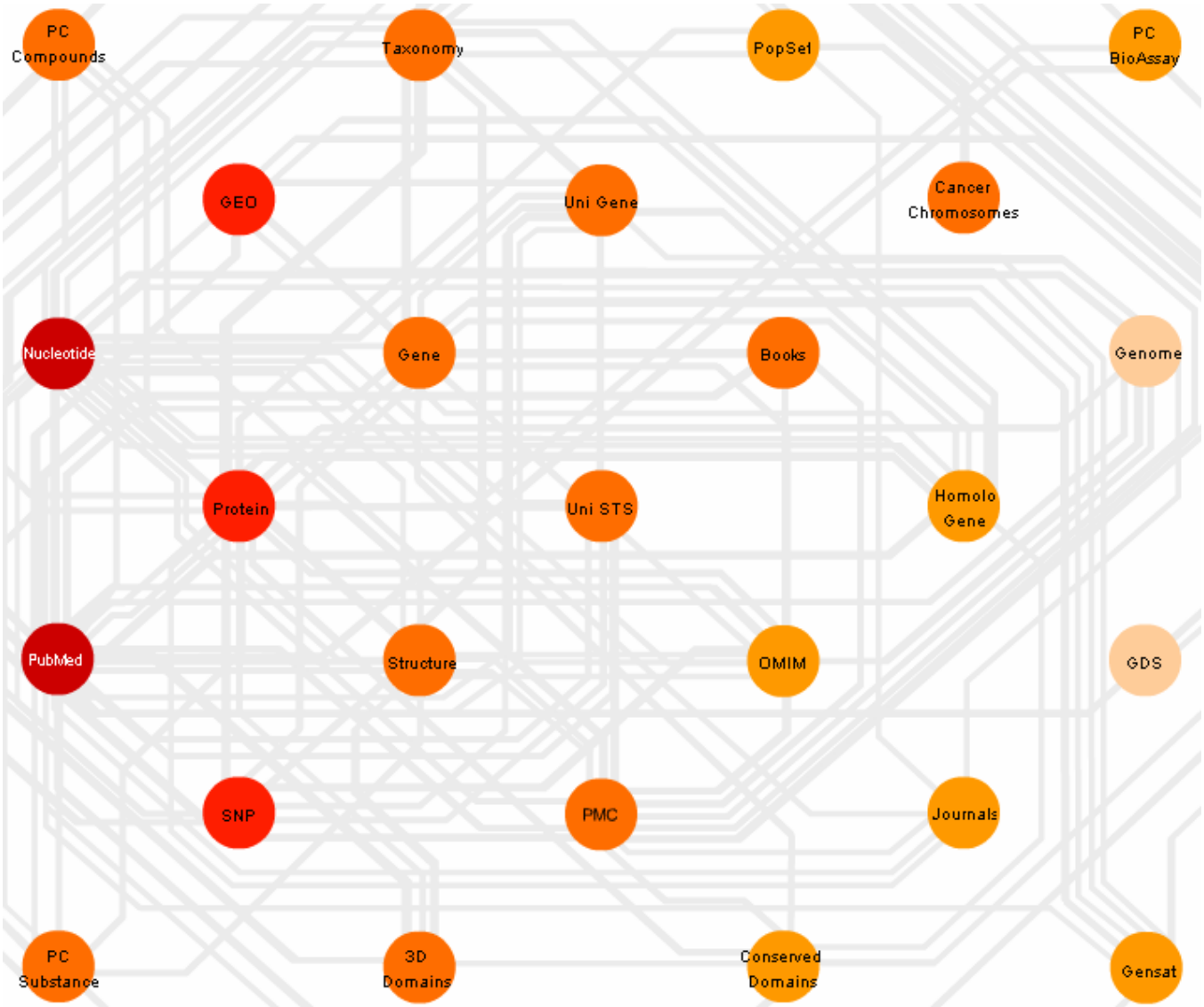
Finalmente, la base de datos **KEGG** se empleará para establecer los mapas genéticos. Necesitaremos:

- Mapas del KEGG dónde se encuentra un gen dado.
- Genes que se encuentran en mismos mapas que un gen dado.

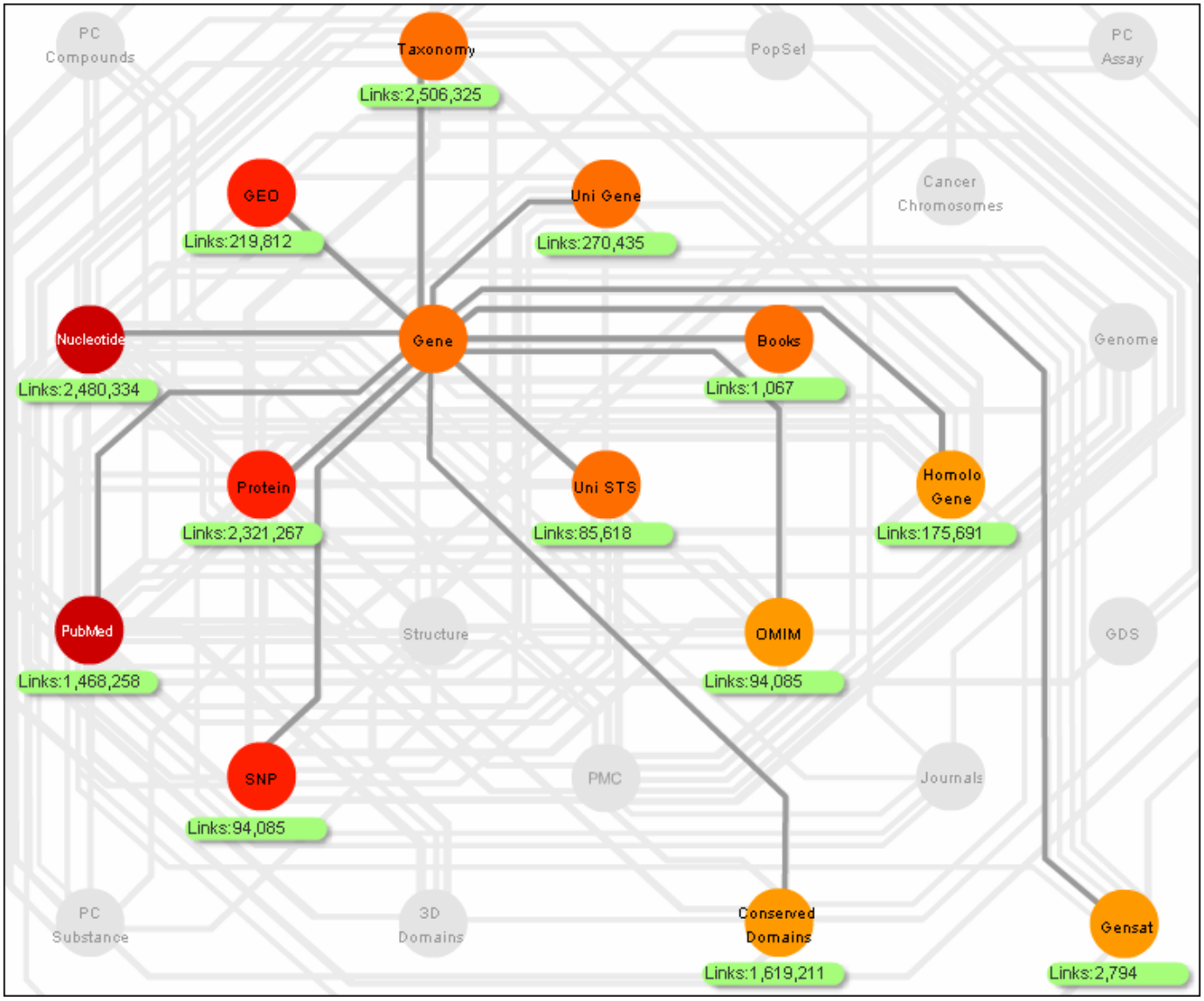
La estructura de Bases de datos del NCBI es mucho más compleja que la que necesitaremos nosotros para la aplicación de cruces on-line.

5.2.1.- Estructura Lógica del NCBI:

Comenzaremos a extraer información de las bases de datos comentadas anteriormente. Para ello investigamos cómo están relacionadas entre sí.



5.2.2.- Base de datos Gene:



Observamos en el gráfico las relaciones existentes entre la base de datos Gene y el resto de bases de datos.

A nosotros nos interesará saber cómo se relaciona con las bases de datos UniGene, PubMed, OMIM y Homologene. Así, a partir de un código de gen, podremos conocer:

- Diferentes tipos de código asociados al mismo gen *(UniGene)*.
- Publicaciones en las que aparece un gen *(PubMed)*.
- Patologías propias de un gen *(OMIM)*.
- Genes homólogos de un gen *(Homologene).*

Analizaremos a continuación las clases y atributos existentes en la base de datos Gene, las relaciones entre las clases y las claves primarias y foráneas que harán de enlace entre el resto de bases de datos.

Del análisis propio realizado sobre la base de datos Gene, tendremos que obtener al menos información que nos revele el GeneID, Nombre, Símbolo, Taxonomía, Localización e Interacciones de un gen.

5.2.2.1.- Identificación de Clases y Atributos:

El objetivo de esta tarea es identificar las responsabilidades y atributos de las clases candidatas que obtengamos en un primer análisis.

Las responsabilidades de una clase definen la funcionalidad de esa clase y están basadas en el estudio de los papeles que desempeñan sus objetos dentro de los diversos casos de uso.

Los atributos de una clase especifican las propiedades de la clase y se identifican por estar implicados en sus responsabilidades. Los tipos de estos atributos son conocidos y especificados en el dominio.

Toda la información contenida en la base de datos Gene se encuentra almacenada y disponible para su descarga en el FTP del NCBI:
[ftp://ftp.ncbi.nih.gov/gene]

La estructura jerárquica del directorio FTP nos proporciona los ficheros con información útil para nuestra aplicación:

ftp://ftp.ncbi.nih.gov
- gene/
  - DATA/
    - gene_info.gz → Información básica de cada gen.
    - gene2go.gz → Gene Ontology para cada gen.
    - gene2refseq.gz → Posición de cada gen en el genoma
  - GeneRIF/
    - interactions.gz → Interacciones de cada gen.

Serán estos ficheros los que hemos de tratar, ya que en ellos se encuentra la base de conocimiento completa de Gene.

Cada uno de estos ficheros serán los que descargarán posteriormente los robots de actualización, descomprimirán y parsearán. De ellos se obtendrán los datos necesarios (que a continuación vamos a decidir) para actualizar la base de datos local.

Mucha información será innecesaria ya que el NCBI trata con datos que a nosotros no son completamente desechables. Veremos pues a continuación un análisis de las clases y atributos existentes en la base de datos Gene. De todos ellos, nos quedaremos con los que proporcionen funcionalidad a nuestro sistema de cruces.

[En negrita marcaremos la clave primaria de cada clase]
[El dominio hace referencia al tipo y rango del atributo de la clase]
[Si un atributo debe ser Not Null, se marcará con un tick verde]
[Especificamos toda la información contenida en la base de datos, posteriormente haremos una criba para quedarnos con aquella que nos sea útil]

Información contenida en el fichero gene_info.gz

gene_info			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
Tax_id	Texto (10)	V	Identificador de la taxonomía de la especie, propio del NCBI.
GeneID	Texto (10)	V	Identificador único de un gen.
Symbol	Texto (20)	V	Símbolo del gen.
LocusTag	Texto (10)		Antiguo valor del gen para la base de datos LocusTag.
Synonyms	Texto (100)		Lista de símbolos no oficiales del gen.
dbXrefs	Texto (100)		Lista de identificadores del gen para otras bases de datos.
Chromosome	Texto (10)		Cromosoma dónde se encuentra situado el gen.
Map_location	Texto (10)		Posición del map location para el gen.
Description	Texto (255)		Nombre descriptivo del gen.
Type Of Gene	Texto (10)		Tipo asignado al gen.
Symbol from nomenclature authority	Texto (100)		Cuando no es "-" indica que el símbolo pertenece a una nomenclatura autorizada.
Full name from nomenclature authority	Texto (100)		Cuando no es "-" indica el estatus del nombre en la nomenclatura autorizada.
Nomenclature status	Texto (100)		Indica el estado del nombre.
Other designations	Texto (100)		Lista de diferentes descripciones que se les ha asignado al gen.
Modification date	Texto (8)		Fecha en la que el gen fue modificado, en el formato YYYYMMDD.

LocusTag hace referencia a la antigua base de datos que empleaba anteriormente el NCBI *(LocusLink)* y que no se actualiza desde el 2005.

Para ver las correspondencias entre esta antigua base de datos y la que emplea actualmente el NCBI *(Entrez Gene)*:

[http://www.ncbi.nlm.nih.gov/entrez/query/static/help/LL2G.html#files]

LocusLink file name	Entrez Gene file name	Comments
LL_tmpl	GeneRIF/generifs_basic.gz	This file extracts the equivalent of the GRIF tag in the LL_tmpl file and reports text of the GeneRIF and the associated PubMed ids.
	DATA/ASN	Files in this directory contain comprehensive extractions from Entrez Gene in ASN.1 format.
LL.out_*	gene_info	GeneID, names, map locations, and database cross-reference.
loc2acc	gene2accession	gene2accession includes the data in gene2refseq above. If you want to convert any accessions into GeneIDs, this one file should suffice.
loc2cit	gene2pubmed	gene2pubmed includes the identifier for the species of the GeneID (*i.e.* the Taxonomy ID).
loc2go	gene2go	GeneID / GO ID / Evidence Code Consolidated summary based on gene_association files from the GO Consortium and Entrez Gene's gene_info file. Differs from loc2go in supplying qualifiers of the GeneID/GO ID relationship, and the GO term itself.
loc2ref	gene2refseq	The file in Entrez Gene does not include information about secondary accessions. This function is now provided from the RefSeq ftp site, as documented in the current release notes: ftp://ftp.ncbi.nlm.nih.gov/refseq/release/release-notes/RefSeq-release#.txt, where # is the value of the current release number.
loc2sts	gene2sts	GeneID / UniSTS marker ID relationship
loc2UG	gene2unigene	GeneID / UniGene cluster relationship
mim2loc	mim2gene	GeneID / MIM number relationship

Tabla de asociaciones de ficheros antiguos pertenecientes a la base de datos LocusLink con los nuevos ficheros de Entrez Gene.

Información contenida en el fichero gene2go.gz: GO terms que han sido asociados con genes en la base de datos Entrez Gene. Se genera mediante el procesamiento de los ficheros del GO contenidos en:
[http://www.geneontology.org/GO.current.annotations.shtml]

gene2go			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
Tax_id	Texto (10)	**V**	Identificador de la taxonomía de la especie, propio del NCBI.
GeneID	Texto (10)	**V**	Identificador único de un gen.
GO_ID	Texto (11)	**V**	Identificador del Gene Ontology, formateado de la forma GO:0000000.
Evidence	Texto (3)	**V**	Código de la evidencia del gen en el fichero de asociaciones (IEA,ND,ISS,TAS).
Qualifier	Texto (3)	**V**	Calificativo de la relación entre el gen y el término GO.
GO_term	Texto (100)	**V**	Nombre del término GO.
PubMed	Texto (62)		Lista de PubMed ids como evidencia de la asociación.
Category	Texto (10)	**V**	Categoría del término GO (Function, Process, Component).

Con esta información conocemos la ontología de un gen dado. En qué funciones celulares se encuentra, en qué procesos actúa y en qué componentes se halla.
Además, mediante una consulta inversa, podemos saber qué genes (GeneID) disponen de misa Función/Proceso/Componente que un gen dado.

Información contenida en el fichero gene2refseq.gz:
Contiene el equivalente al fichero:
[ftp://ftp.ncbi.nih.gov/refseq/LocusLink/loc2ref]

gene2refseq			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
Tax_id	Texto (10)	**V**	Identificador de la taxonomía de la especie, propio del NCBI.
GeneID	Texto (10)	**V**	Identificador único de un gen.
Protein GI	Texto (10)	**V**	GI para un Protein accession.
Start	Int (10)	**V**	Posición de comienzo del gen en el genoma.
End	Int (10)	**V**	Posición de fin del gen en el genoma.
Orientation	Texto (10)	**V**	Orientación del gen en el genoma.

Este fichero proporciona la posición de un gen dentro del genoma. Con esta información podemos conocer los genes vecinos a este gen, tomando el inicio y seleccionando aquellos genes que comiencen cerca de éste.

Información contenida en el fichero interactions.gz:
Descripción de las interacciones entre dos genes y de su resultado.
Si los dos integrantes de la interacción son genes, el par sólo se reporta una vez, usando el convenio de que el gen con menor GeneID aparece como *First Interactant*
Este fichero incluye además los datos contenidos en el fichero *hiv_interactions.gz.*

interactions			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
First interactanct Tax_id	Texto (10)	**V**	Identificador de la taxonomía de la especie, propio del NCBI.
First interactant ID	Texto (10)	**V**	Identificador único del primer gen de la interacción (tiene menor GeneID).
First interactant name	Texto (255)		Nombre del primer gen en la interacción.
Second interactanct Tax_id	Texto (10)	**V**	Identificador de la taxonomía de la especie, propio del NCBI.
Second interactant ID	Texto (10)	**V**	Identificador único del segundo gen de la interacción (tiene mayor GeneID).
Second interactant name	Texto (255)		Nombre del segundo gen en la interacción.

Second interactant type	Texto (10)		Database dónde podrá ser encontrado el interactant ID. Si es un gen, este campo será "GeneID".
Complex ID	Texto (10)		Identificador para el resultado de la interacción.
Complex name	Texto (255)		Nombre del complejo resultado.
Complex type	Texto (10)		Base de datos dónde se halla el Complex ID.
PubMed	Texto (62)		Lista de PubMed ids como evidencia de la interacción.
GeneRif_text	Texto (255)	**V**	Nombre de la interacción.
Interaction ID	Texto (10)		Identificador de la interacción.
Interaction type	Texto (10)		Base de datos dónde encontrar el Interaction ID.

A partir de esta información se puede conocer las interacciones existentes de un gen dado. Basta con buscar en ambos identificadores de interactuadores el GeneID del gen dado. De esta forma, se obtienen los genes con menor GeneID que interactúan con él (si el GeneID del gen dado se encuentra en el *Second Interactant ID*) y los genes con mayor GeneID que interactúan con él (si el GeneID del gen dado se encuentra en el *First Interactant ID*).

Además podemos conocer el nombre de dicha interacción, así como una referencia en los documentos publicados de PubMed.

5.2.2.2.- Descripción de Relaciones:

MENOR INTERACTUADOR			
Grado	**Cardinalidad**	**ENTIDADES**	**DESCRIPCIÓN**
2	1:N	**gene_info, interactions**	Un gen puede aparecer cómo menor GeneID en una serie de interacciones con otros genes. Ese gen que interacciona con diversos genes sólo tendrá una entrada en la tabla de gene_info, con su información principal.
La clave primaria de *gene_info* (**GeneID**), hará de clave foránea en *interactions* (**First Interactant ID**) de manera que enlazamos un gen con sus interacciones.			

MAYOR INTERACTUADOR			
Grado	**Cardinalidad**	**ENTIDADES**	**DESCRIPCIÓN**
2	1:N	**gene_info, interactions**	Un gen puede aparecer cómo mayor GeneID en una serie de interacciones con otros genes. Ese gen que interacciona con diversos genes sólo tendrá una entrada en la tabla de gene_info, con su información principal.
La clave primaria de *gene_info* (**GeneID**), hará de clave foránea en *interactions* (**Second Interactant ID**) de manera que enlazamos un gen con sus interacciones.			

POSICIÓN EN GENOMA			
Grado	**Cardinalidad**	**ENTIDADES**	**DESCRIPCIÓN**
2	1:1	**gene_info, gene2refseq**	Un gen pertenece a un genoma y dispone de una posición de inicio y de fin en dicho genoma.
La clave primaria de *gene_info* (**GeneID**), hará de clave foránea en *gene2refseq* (**GeneID**) de manera que enlazamos un gen con su posición de inicio y fin en el genoma.			

ONTOLOGÍA			
Grado	**Cardinalidad**	**ENTIDADES**	**DESCRIPCIÓN**
2	1:N	**gene_info, gene2go**	Un gen dispone de una serie de GO_ID, cada uno de ellos diferenciando las diferentes funciones, componentes y/o procedimientos celulares en los que interviene dicho gen.
La clave primaria de *gene_info* (**GeneID**), hará de clave foránea en *gene2go* (**GeneID**) de manera que enlazamos un gen con la totalidad de su ontología (Funciones, Componentes y Procesos en los que interviene).			

Estas relaciones son las existentes entre las clases relevantes de la base de datos de Gene.

La clase central, cómo puede observarse, es la gene_info, conteniendo ésta la información básica de cada gen.
A partir del GeneID se accede a las diversas actividades de las que participa este gen en el entorno celular.

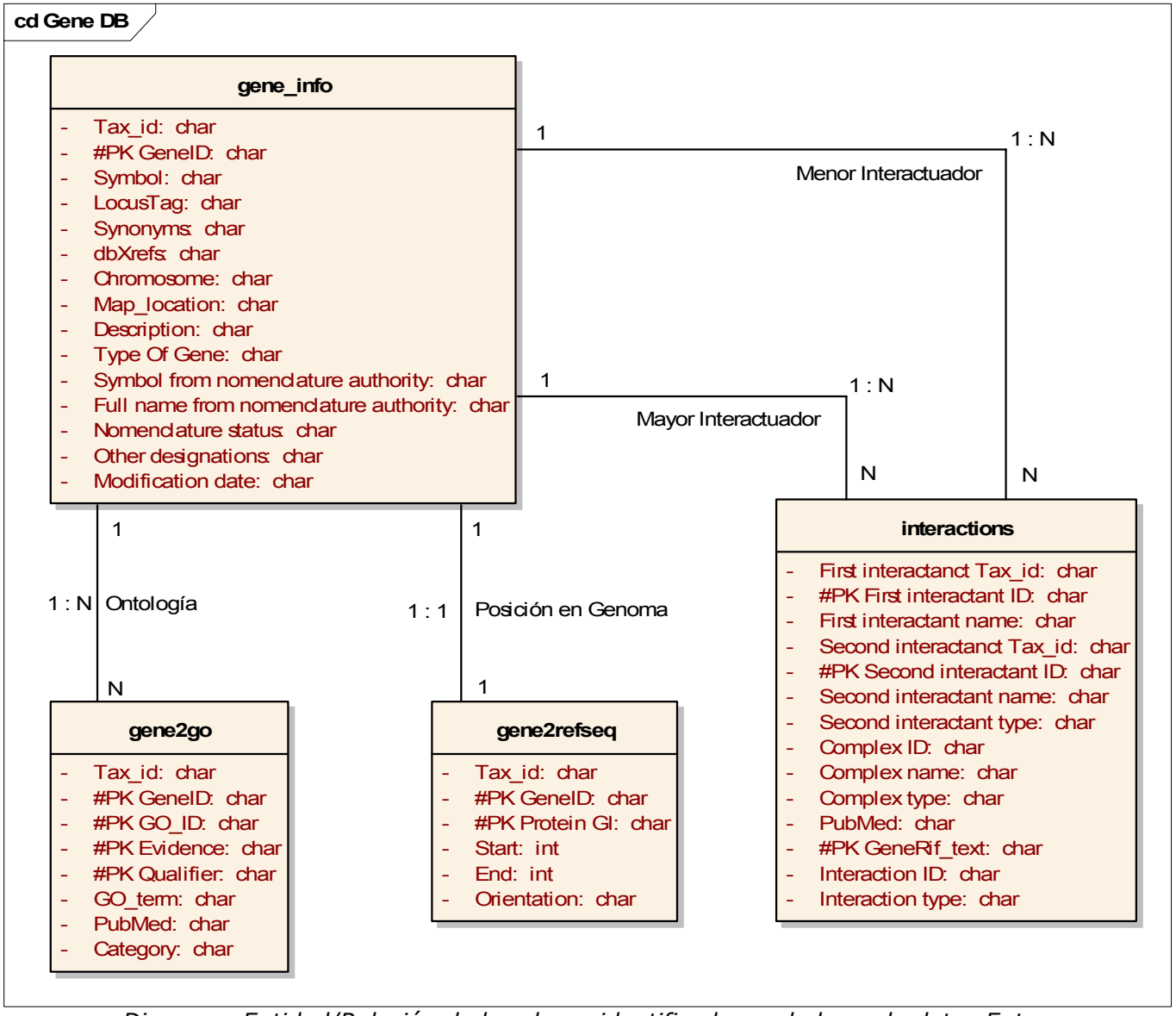


Diagrama Entidad/Relación de las clases identificadas en la base de datos Entrez Gene y útiles para nuestra aplicación.

5.2.2.3.- Minería de datos, selección de clases y atributos:

Hemos analizado en el punto anterior las clases con información relevante para poder llevar a cabo nuestras consultas a bases de datos externas. De toda esta información, haremos una selección de sólo aquellas clases y atributos que nos serán útiles para el posterior proceso de creación y volcado de la base de datos en local.

Así pues, en nuestro diagrama lógico sólo dispondremos de las clases, con sus correspondientes atributos, que proporcionan la información básica y necesaria para las consultas que realmente nos interesan.
El resto de información existente en la base de datos *Entrez Gene* del NCBI será desechada.

Los robots de actualización de la base de datos local serán los encargados de seleccionar la información útil de los ficheros de bases de datos y actualizarla en local *(Ver documento de Diseño)*.

A la hora de poder realizar consultas y cruces on-line, la información que nos interesará con respecto a la base de datos de **Gene** será la siguiente:

- Información básica del gen: GeneID, Símbolo, Nombre, Cromosoma en el que se encuentra y Taxonomía.
- Interacciones: genes con los que interacciona un gen.
- Gene Ontology: función, proceso y componente de un gen.
- Genes que están en misma función, proceso y/o componente que un gen.
- Aproximate Location: obtener genes vecinos a un gen dado.

- Para obtener la información básica del gen utilizaremos los siguientes datos de *gene_info*:

gene_info			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
Tax_id	Texto (10)	**V**	Identificador de la taxonomía de la especie, propio del NCBI.
GeneID	Texto (10)	**V**	Identificador único de un gen.
Symbol	Texto (20)	**V**	Símbolo del gen.
LocusTag	Texto (10)		Antiguo valor del gen para la base de datos LocusTag.
Synonyms	Texto (100)		Lista de símbolos no oficiales del gen.
dbXrefs	Texto (100)		Lista de identificadores del gen para otras bases de datos.
Chromosome	Texto (10)		Cromosoma dónde se encuentra situado el gen.
Map_location	Texto (10)		Posición del map location para el gen.
Description	Texto (255)		Nombre descriptivo del gen.
Type Of Gene	Texto (10)		Tipo asignado al gen.
Symbol from	Texto (100)		Cuando no es "-" indica que el

nomenclature authority			símbolo pertenece a una nomenclatura autorizada.
Full name from nomenclature authority	Texto (100)		Cuando no es "-" indica el estatus del nombre en la nomenclatura autorizada.
Nomenclature status	Texto (100)		Indica el estado del nombre.
Other designations	Texto (100)		Lista de diferentes descripciones que se les ha asignado al gen.
Modification date	Texto (8)		Fecha en la que el gen fue modificado, en el formato YYYYMMDD.

Con Tax_id, GeneID, Symbol y Description tenemos la suficiente información de un gen determinado:

- GeneID → Identificador único para el gen.
- Symbol → Símbolo del gen.
- Description → Nombre del gen.
- Tax_id → Identificador de taxonomía de especie del gen.

Nuestra clase final para la información básica del gen será la siguiente:

gene_info			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
Tax_id	Texto (10)	**V**	Identificador de la taxonomía de la especie, propio del NCBI.
GeneID	Texto (10)	**V**	Identificador único de un gen.
Symbol	Texto (20)	**V**	Símbolo del gen.
Description	Texto (255)		Nombre descriptivo del gen.

- Para conocer los genes con los que interactúa un gen, así cómo el nombre que recibe dicha interacción, deberemos quedarnos con los siguientes atributos de *interactions*.

interactions			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
First interactanct Tax_id	Texto (10)	**V**	Identificador de la taxonomía de la especie, propio del NCBI.
First interactant ID	Texto (10)	**V**	Identificador único del primer gen de la interacción (tiene menor GeneID).
First interactant name	Texto (255)		Nombre del primer gen en la interacción.
Second interactanct Tax_id	Texto (10)	**V**	Identificador de la taxonomía de la especie, propio del NCBI.
Second interactant ID	Texto (10)	**V**	Identificador único del segundo gen de la interacción (tiene mayor GeneID).
Second interactant name	Texto (255)		Nombre del segundo gen en la interacción.
Second interactant type	Texto (10)		Database dónde podrá ser encontrado el interactant ID. Si es un gen, este campo será "GeneID".
Complex ID	Texto (10)		Identificador para el resultado de la interacción.
Complex name	Texto (255)		Nombre del complejo resultado.
Complex type	Texto (10)		Base de datos dónde se halla el Complex ID.
PubMed	Texto (62)		Lista de PubMed ids como evidencia de la interacción.
GeneRif_text	Texto (255)	**V**	Nombre de la interacción.
Interaction ID	Texto (10)		Identificador de la interacción.
Interaction type	Texto (10)		Base de datos dónde encontrar el Interaction ID.

Nos bastará con el Tax_id y GeneID's de los genes que participan en la interacción, así cómo con la lista de entradas en el PubMed como evidencia de la interacción y en nombre que recibe.

Para conocer los genes que interactúan con un gen dado:

- Tomaremos aquellos con menor GeneID que interaccionen con él.
- Tomaremos aquellos con mayor GeneID que interaccionen con él.
- En ambos casos, existirá un nombre para la interacción y una lista de publicaciones.

Para una mayor comprensión, emplearemos nuestros propios nombres para los atributos en la base de datos local:

Nombre de la clase: gene2interactions en lugar de interactions.
First interactant Tax_id ←→ Tax_id
First interactant ID ←→ GeneID
Second interactant ID ←→ interactantID

Nuestra clase final para las interacciones entre genes será la siguiente:

gene2interaction			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
Tax_id	Texto (10)	**V**	Identificador de la taxonomía de la especie, propio del NCBI.
GeneID	Texto (10)	**V**	Identificador único del primer gen de la interacción (tiene menor GeneID).
interactantID	Texto (10)	**V**	Identificador único del segundo gen de la interacción (tiene mayor GeneID).
PubMed	Texto (62)		Lista de PubMed ids como evidencia de la interacción.
GeneRif_text	Texto (255)	**V**	Nombre de la interacción.

Hemos elimiinado los nombres de cada gen que interacciona porque puede obtenerse mediante su GeneID consultando en la tabla gene_info.
Además tampoco necesitamos el resultado de la interacción, ya que no es un requisito que vaya a hacer falta en nuestra aplicación.

- Para conocer el genoma en el que se encuentra un gen, así cómo su comienzo de manera que podamos indexar sus genes vecinos nos quedaremos con los siguientes atributos de *gene2refseq:*

gene2refseq			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
Tax_id	Texto (10)	**V**	Identificador de la taxonomía de la especie, propio del NCBI.
GeneID	Texto (10)	**V**	Identificador único de un gen.
Protein GI	Texto (10)	**V**	GI para un Protein accession.
Start	Int (10)	**V**	Posición de comienzo del gen en el genoma.
End	Int (10)	**V**	Posición de fin del gen en el genoma.
Orientation	Texto (10)	**V**	Orientación del gen en el genoma.

Nos bastará con el GeneID y Protein GI para conocer el genoma en el que se encuentra un gen. Además, conociendo el valor dónde comienza dicho gen, podremos consultar en la tabla aquellos que, estando en el mismo genoma, comienzan en una posición cercana a la suya.

Así pues, con GeneID (clave foránea en *gene_info*), Protein GI y Start conseguimos la funcionalidad requerida para conocer los vecinos de un gen.

Para una mayor comprensión, emplearemos nuestros propios nombres para los atributos en la base de datos local:

Nombre de la clase: gene2map en lugar de gene2refseq.
Protein GI ←→ GI

Nuestra clase final para conocer los genes vecinos de un determinado gen será la siguiente:

gene2map			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
GeneID	Texto (10)	**V**	Identificador único de un gen.
GI	Texto (10)	**V**	GI para un Protein accession.
Start	Int (10)	**V**	Posición de comienzo del gen en el genoma.

No necesitamos conocer el Tax_id ya que lo encontramos asociado al mismo GeneID en la tabla gene_info.
La posición de fin no será necesaria, para conocer los vecinos de un gen nos basta con buscar aquellos que comiencen por una posición cercana a la suya.

- Para conocer la ontología de un gen dado, así cómo aquellos genes que disponen de misma ontología que él, nos quedaremos con los siguientes atributos de *gene2go:*

gene2go			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
Tax_id	Texto (10)	**V**	Identificador de la taxonomía de la especie, propio del NCBI.
GeneID	Texto (10)	**V**	Identificador único de un gen.
GO_ID	Texto (11)	**V**	Identificador del Gene Ontology, formateado de la forma GO:0000000.
Evidence	Texto (3)	**V**	Código de la evidencia del gen en el fichero de asociaciones (IEA,ND,ISS,TAS).
Qualifier	Texto (3)	**V**	Calificativo de la relación entre el gen y el término GO.
GO_term	Texto (100)	**V**	Nombre del término GO.
PubMed	Texto (62)		Lista de PubMed ids como evidencia de la asociación.
Category	Texto (10)	**V**	Categoría del término GO (Function, Process, Component).

En este caso, todos los atributos no son necesarios, ya que GeneID, GO_ID, Evidence y Qualifier hacen de clave primaria para diferenciar entre los diferentes términos GO de un gen.

- GO_term nos proporciona el nombre informativo que describirá al término en la pantalla de resultados.
- Category servirá como filtro de búsqueda, de manera que podamos obtener genes que aparezcan en misma Function, Process y/o Component que un gen dado.

La estructura lógica final de la base de datos Gene en nuestro servidor local será como la que sigue:

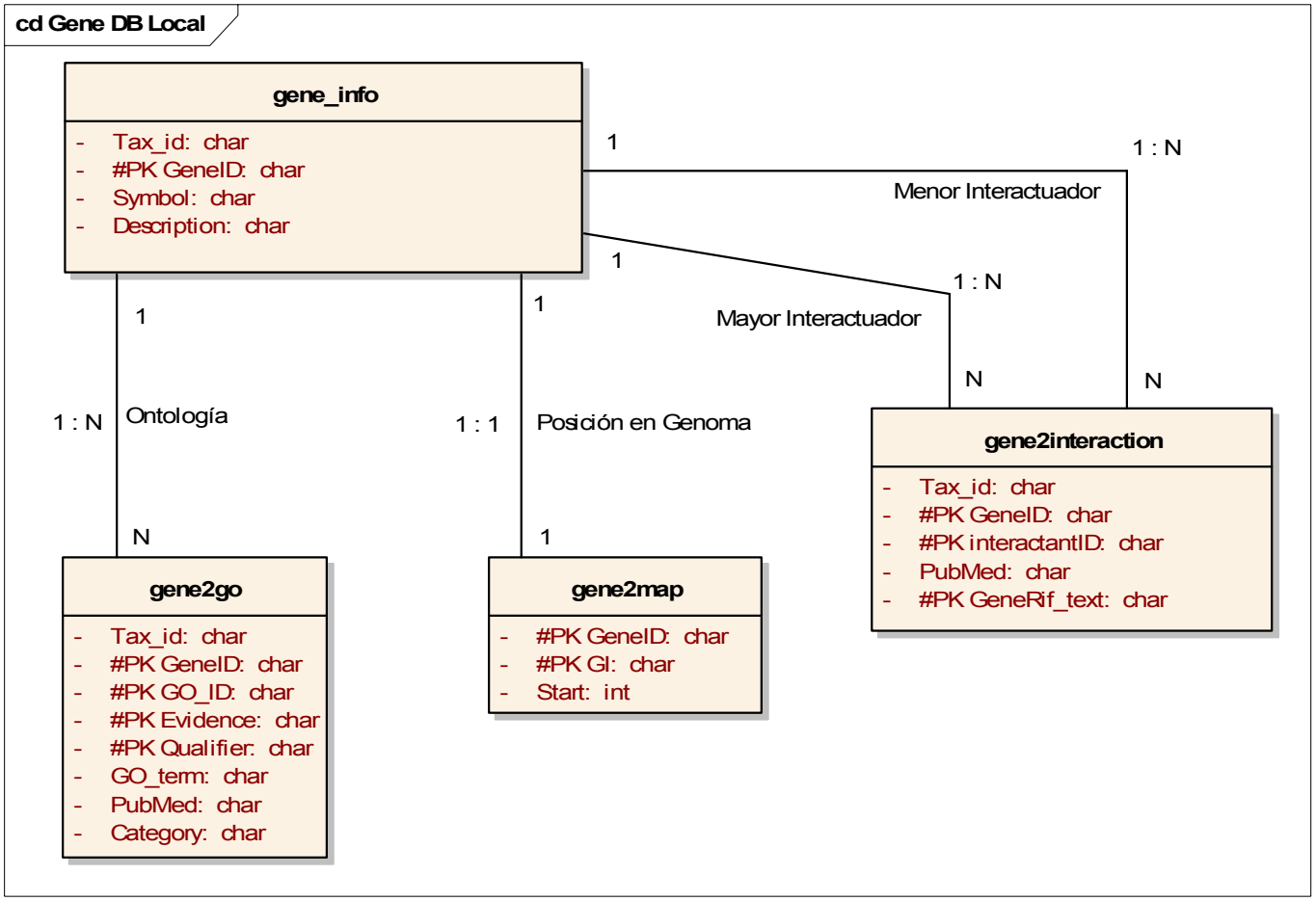


Diagrama Entidad/Relación de las clases finales en nuestra base de datos local, con información extraída de Entrez Gene.

En este diagrama se recoge toda la información necesaria para llevar a cabo la funcionalidad relacionada con la base de datos **Gene** que implementará nuestra aplicación de cruce de información.

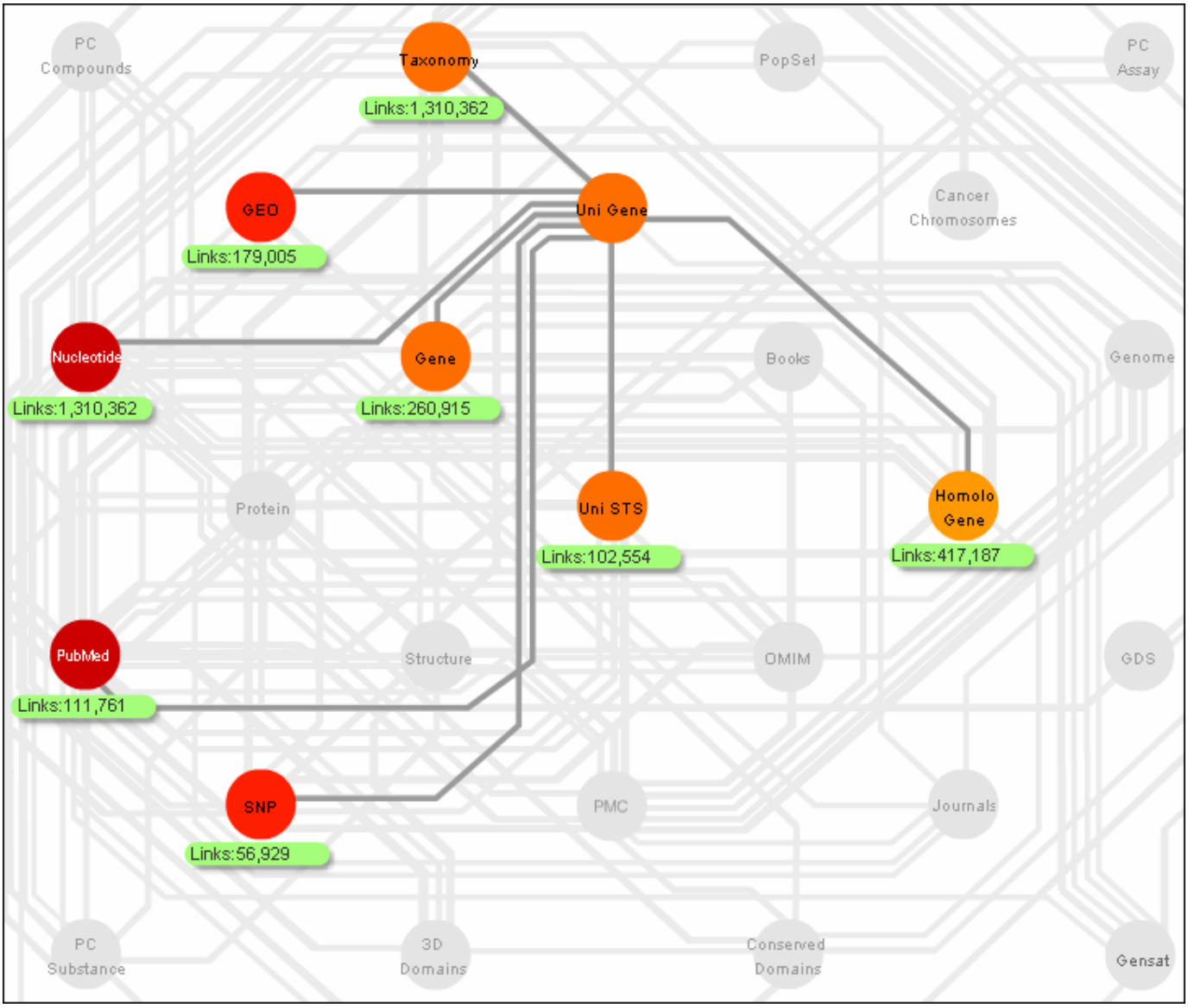
Las clases y atributos seleccionados son los mínimos indispensables, de manera que no exista información duplicada, que la base de datos sea coherente y que todas las operaciones puedan llevarse a cabo de manera rápida y eficaz.

El robot que actualiza las bases de datos locales será el encargado de descargarse los ficheros mencionados anteriormente, obtener de ellos la información relevante indicada en este diagrama y actualizar la base de datos local.

Así pues, hemos conseguido, haciendo minería de datos sobre la compleja e inmensa base de datos *Entrez Gene*, los atributos necesarios que garanticen la funcionalidad requerida:

- Información básica del gen: GeneID, Símbolo, Nombre, Cromosoma en el que se encuentra y Taxonomía.
- Interacciones: genes con los que interacciona un gen.
- Gene Ontology: función, proceso y componente de un gen.
- Genes que están en misma función, proceso y/o componente que un gen.
- Aproximate Location: obtener genes vecinos a un gen dado.

5.2.3.- Base de datos UniGene:



Observamos en el gráfico las relaciones existentes entre la base de datos UniGene y el resto de bases de datos.

A nosotros nos interesará saber cómo se relaciona con la base de datos Gene, únicamente. La única funcionalidad que nos interesará de la base de datos UniGene será la unificación de identificadores de gen.

Así pues, tendremos que disponer de la información necesaria para que, a partir de diversos tipos de códigos de gen con los que podemos encontrarnos, unificarlos y conseguir su código GeneID asociado. De esta forma, trabajaremos siempre con GeneID's y no con un sinfín de posibles códigos de gen existentes sin unificar.

En las microarrays podemos encontrar diversos formatos de códigos de gen:

- Indicados como taxonomía de especie y número de gen: Hs.112 (Homo Sapiens 112).
- Indicados como número de secuencia dentro del cromosoma (Tras el campo 3' o 5', dependiendo de la orientación del gen).

5.2.3.1.- Identificación de Clases y Atributos:

Toda la información contenida en la base de datos UniGene se encuentra almacenada y disponible para su descarga en el FTP del NCBI:
[ftp://ftp.ncbi.nih.gov/repository/UniGene]

La estructura jerárquica del directorio FTP nos proporciona los ficheros con información útil para nuestra aplicación:

ftp://ftp.ncbi.nih.gov/
repository/
UniGene/
Acyrthosiphon_pisum
Aedes_aegypti
Anopheles_gambiae
….
Zea_mays

Existe un directorio para cada especie conocida. Dentro de estos directorios nos interesan básicamente dos tipos de ficheros:

- XX.seq.all.gz → fichero que contiene las correspondencias de secuencias de genes a códigos de UniGene (del tipo Hs.123).
- XX.data.gz → fichero que contiene las correspondencias de códigos de UniGene a GeneID's.

XX es el identificador de la especie en cada directorio.
El proceso natural para obtener el GeneID sería:

- Si disponemos de un código de secuencia de gen, convertiríamos éste al código Unigene a través de la correspondencia dada en el fichero XX.seq.all. Una vez obtenido el código UniGene (son del tipo Hs.112), bastaría con hallar su correspondencia con el GeneID en el fichero XX.data.
- Si por el contrario, disponemos de un código UniGene cómo entrada del gen, tan sólo habría que obtener la correspondencia con GeneID del fichero XX.data.

Con tódo esto, necesitaremos tratar los dos ficheros indicados, para todas las especies existentes. En la unión de todos estos ficheros se encuentra la base de conocimiento necesaria para realizar la unificación de los dos tipos de códigos citados anteriormente a GeneID.

Cada uno de estos ficheros serán los que descargarán posteriormente los robots de actualización, descomprimirán y parsearán. De ellos se obtendrán los datos necesarios (que a continuación vamos a decidir) para actualizar la base de datos local.

El proceso puede ser costoso en tiempo, puesto que existen gran cantidad de especies y los ficheros a tratar son del orden de GigaBytes.

Por el contrario que en el NCBI, nosotros dispondremos todos los datos de todas las especies unificados en dos tablas, en lugar de una para cada especie:

- Una para la correspondencia: Código de secuencia ←→ Código UniGene.
- Otra para la correspondencia: Código UniGene ←→ GeneID

[En negrita marcaremos la clave primaria de cada clase]
[El dominio hace referencia al tipo y rango del atributo de la clase]
[Si un atributo debe ser Not Null, se marcará con un tick verde]
[Especificamos toda la información contenida en la base de datos, posteriormente haremos una criba para quedarnos con aquella que nos sea útil]

Información contenida en el fichero xx.data

data			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
ID	Texto (15)	V	UniGene Cluster ID
Title	Texto (255)	V	Nombre del UniGene Cluster.
Gene	Texto (15)		Símbolo del gen.
Cytoband	Texto (10)		Banda citológica.
Express	Texto (10)		Tejidos que originan el gen en el cluster.
Restr_expr	Texto (10)		Tejido que aparece con más frecuencia en el gen.
Gnm_terminus	Texto (10)		Confirmación de aparición de término 3’
GeneID	Texto (15)	V	Identifiación de GeneID para la base de datos *Entrez Gene*.
Locuslink	Texto (10)		Identificador Locuslink, desfasado a favor del uso de GeneID.
STS	Texto (10)		Identificador para la base de datos UNISTS del NCBI.
TXMAP	Texto (10)		Intervalo del mapa del gen.
Protsim	Texto (10)		Datos similares proteínicos para la secuencia.
Scount	Int (3)		Número de secuencias en el cluster.
Sequence	Texto (100)		Información de la secuencia.

Toda la información establece los parámetros para un cluster determinado de UniGene. Este cluster contendrá una serie de secuencias, que se pueden obtener a partir del otro fichero.

De esta tabla nos interesará únicamente los valores que nos permitan convertir un código UniGene a otro GeneID, de manera que trabajemos siempre con el resto de datos de Gene empleando GeneID’s.

La conversión a GeneID's será el primer paso a la hora de tratar una microarray. Éstas vienen dadas generalmente empleando códigos de secuencia (del tipo 3' y 5') y nuestra principal función será, a partir de las relaciones que veremos a continuación, convertir todos los genes de la microarray a su correspondiente GeneID.
Información contenida en el fichero xx.seq.all

seq.all			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
ID	Texto (15)	**V**	UniGene Cluster ID.
CDS	Texto (15)	**V**	Coding Sequence.
UniGene sequence	Texto (15)	**V**	Código de secuencia UniGene.

Este fichero, existente para cada especie, contiene la relación entre códigos de secuencia UniGene (del tipo 3' y 5') y códigos de cluster UniGene (del tipo Hs.123).

El UniGene sequence no cambia en las sucesivas actualizaciones que realiza el NCBI, asegurando que siempre permanecerán iguales. Lo que si puede cambiar es al cluster que permanezca este número de secuencia.

Con estos datos ya podemos unificar los códigos de genes de nuestras microarrays. Partiendo de un código *UniGene sequence*, podemos obtener el cluster al que pertenece (*UniGene Cluster ID*). A través de este cluster, y consultando en el fichero XX.data, podemos otener el *GeneID* del gen.

5.2.3.2.- Descripción de Relaciones:

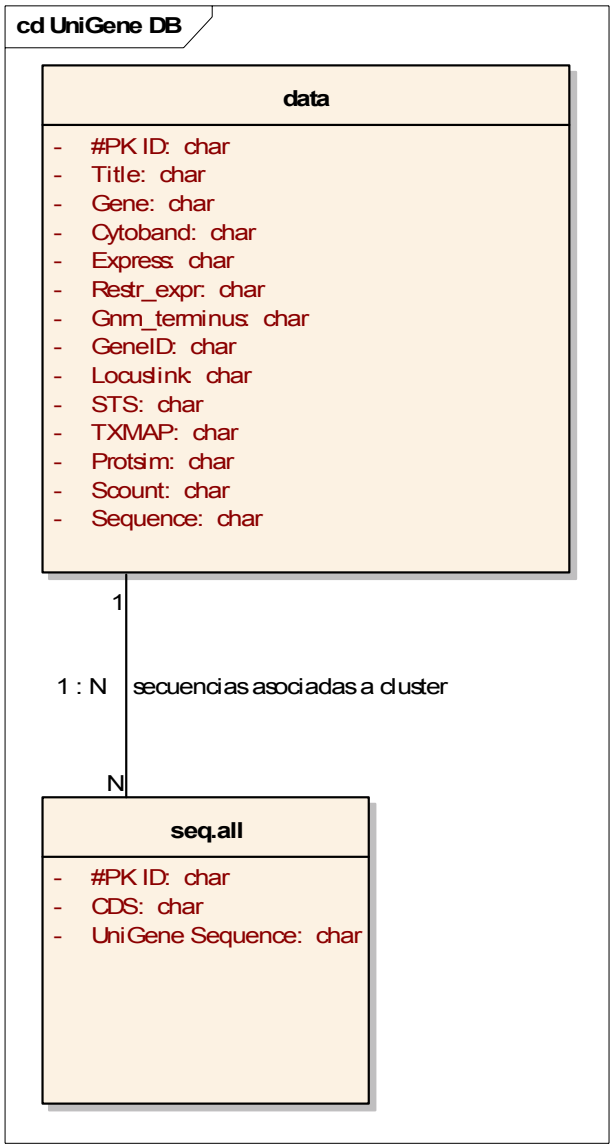
secuencias asociadas a cluster			
Grado	**Cardinalidad**	**ENTIDADES**	**DESCRIPCIÓN**
2	1:N	**data, seq.all**	Con esta relación, se obtienen la totalidad de secuencias asociadas a un código UniGene. En la tabla data tenemos una entrada para cada código UniGene, con su GeneID correspondiente. En la tabla seq.all tenemos una entrada de código UniGene para cada secuencia que pertenece a dicho cluster UniGene.
La clave primaria de *data* (**ID**), hará de clave foránea en *seq.all* (**ID**) de manera que enlazamos un cluster UniGene con sus secuencias asociadas.			

Con esta relación tenemos construido el sistema para generar GeneID's a partir de códigos UniGene cluster (del tipo Hs.123) o de códigos UniGene sequences (del tipo 3' o 5').

En la tabla seq.all guardamos la totalidad de secuencias asociadas a un cluster UniGene.
En la tabla data guardamos el código GeneID asociado a dicho cluster UniGene.

Mostramos a continuación el modelo lógico que nos interesa de la base de datos UniGene del NCBI.

Debemos tener en cuenta que el modelo aquí mostrado se repite para todas y cada unas de las especies existentes en la base de datos y que pueden encontrarse en el directorio de FTP indicado previamente:



En nuestro caso, todos los datos repartidos por especies en los diferentes directorios los unificaremos en sólo dos tablas parecidas a las dos indicadas sobre estas líneas.

Haremos una minería de datos sobre los ficheros XX.seq.all y XX.data de cada especie y, de todos ellos, obtendremos los datos interesantes para nuestra operación de unificación de códigos.

Habremos conseguido así lograr trabajar con sólo códigos GeneID's, códigos que se emplean en la base de datos de Gene previamente diseñada.

5.2.3.3.- Minería de datos, selección de clases y atributos:

Hemos analizado en el punto anterior las clases con información relevante para poder llevar a cabo nuestra unificación de códigos de gen. De toda esta información, haremos una selección de sólo aquellas clases y atributos que nos serán útiles para el posterior proceso de creación y volcado de la base de datos en local.

Así pues, en nuestro diagrama lógico sólo dispondremos de las clases, con sus correspondientes atributos, que proporcionan la información básica y necesaria para las consultas que realmente nos interesan.
El resto de información existente en la base de datos *Entrez UniGene* del NCBI será desechada.

Los robots de actualización de la base de datos local serán los encargados de seleccionar la información útil de todos los ficheros de cada especie y actualizarla en local *(Ver documento de Diseño)*.

Para conseguir todos los posibles códigos de gen que aparezcan en la microrarray, la información que nos interesará con respecto a la base de datos de **UniGene** será la siguiente:

- Correspondencia de código de secuencia UniGene con UniGene cluster.
- Correspondencia de código UniGene cluster con GeneID.

- Para la correspondencia de código de secuencia con Unigene cluster, usaremos la siguiente información de *seq.all*:

seq.all			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
ID	Texto (15)	**V**	UniGene Cluster ID.
CDS	Texto (15)	**V**	Coding Sequence.
UniGene sequence	Texto (15)	**V**	Código de secuencia UniGene.

Nos bastará con el campo de código de secuencia y el campo de código de cluser UniGene.

Para una correcta comprensión, cambiaremos los nombres en nuestro modelo, pasándose a llamar:

Seq.all → sequences.
ID → UniGene_cluster.
UniGene sequence → Sequence.

Nuestra clase en el modelo de la base de datos será pues:

sequences			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
UniGene_cluster	Texto (15)	**V**	UniGene Cluster ID.
Sequence	Texto (15)	**V**	Código de secuencia UniGene.

- Para la correspondencia de código Unigene cluster con GeneID, usaremos la siguiente información de *data*:

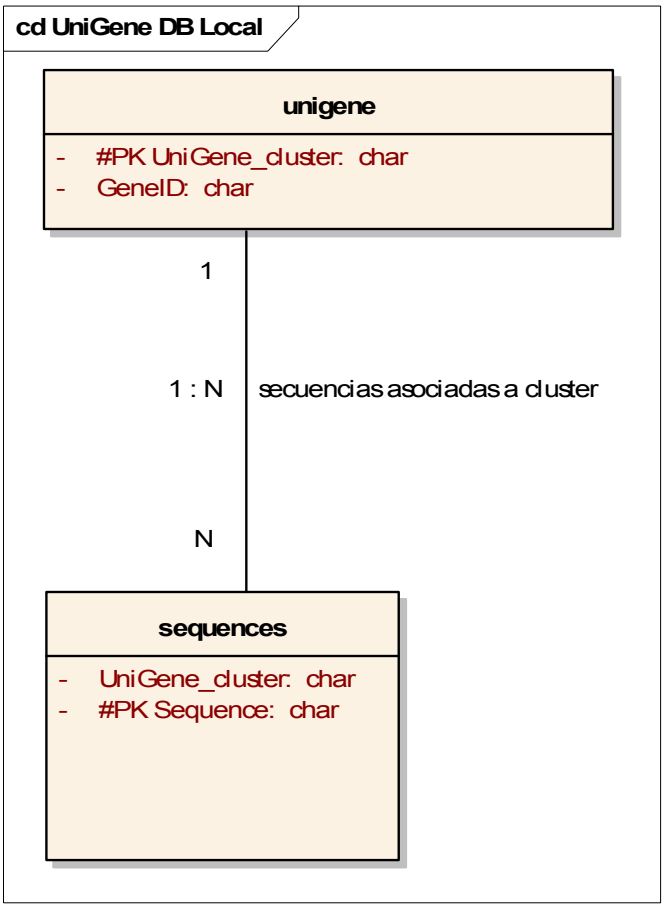
data			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
ID	Texto (15)	**V**	UniGene Cluster ID
Title	Texto (255)	**V**	Nombre del UniGene Cluster.
Gene	Texto (15)		Símbolo del gen.
Cytoband	Texto (10)		Banda citológica.
Express	Texto (10)		Tejidos que originan el gen en el cluster.
Restr_expr	Texto (10)		Tejido que aparece con más frecuencia en el gen.
Gnm_terminus	Texto (10)		Confirmación de aparición de término 3’
GeneID	Texto (15)	**V**	Identifiación de GeneID para la base de datos *Entrez Gene*.
Locuslink	Texto (10)		Identificador Locuslink, desfasado a favor del uso de GeneID.
STS	Texto (10)		Identificador para la base de datos UNISTS del NCBI.
TXMAP	Texto (10)		Intervalo del mapa del gen.
Protsim	Texto (10)		Datos similares proteínicos para la secuencia.
Scount	Int (3)		Número de secuencias en el cluster.
Sequence	Texto (100)		Información de la secuencia.

Vemos que con sólo ID y GeneID ya tenemos la correspondencia de código UniGene Cluster con su GeneID asociado.

Así pues, con estos dos campos y modificando el nombre de los mismos para una mayor comprensión, dispondremos de nuestra clase con sus atributos necesarios en nuestro modelo:

unigene			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
UniGene_cluster	Texto (15)	**V**	UniGene Cluster ID
GeneID	Texto (15)	**V**	Identifiación de GeneID para la base de datos *Entrez Gene*.

Vemos que hemos cambiado *ID* por *UniGene_cluster* y el nombre de la clase pasa a llamarse *unigene* en lugar de *data* (que era demasiado genérico).



Clase sequences:

- Dado un código de secuencia, obtenemos su UniGene_cluster asociado.

Clase unigene:

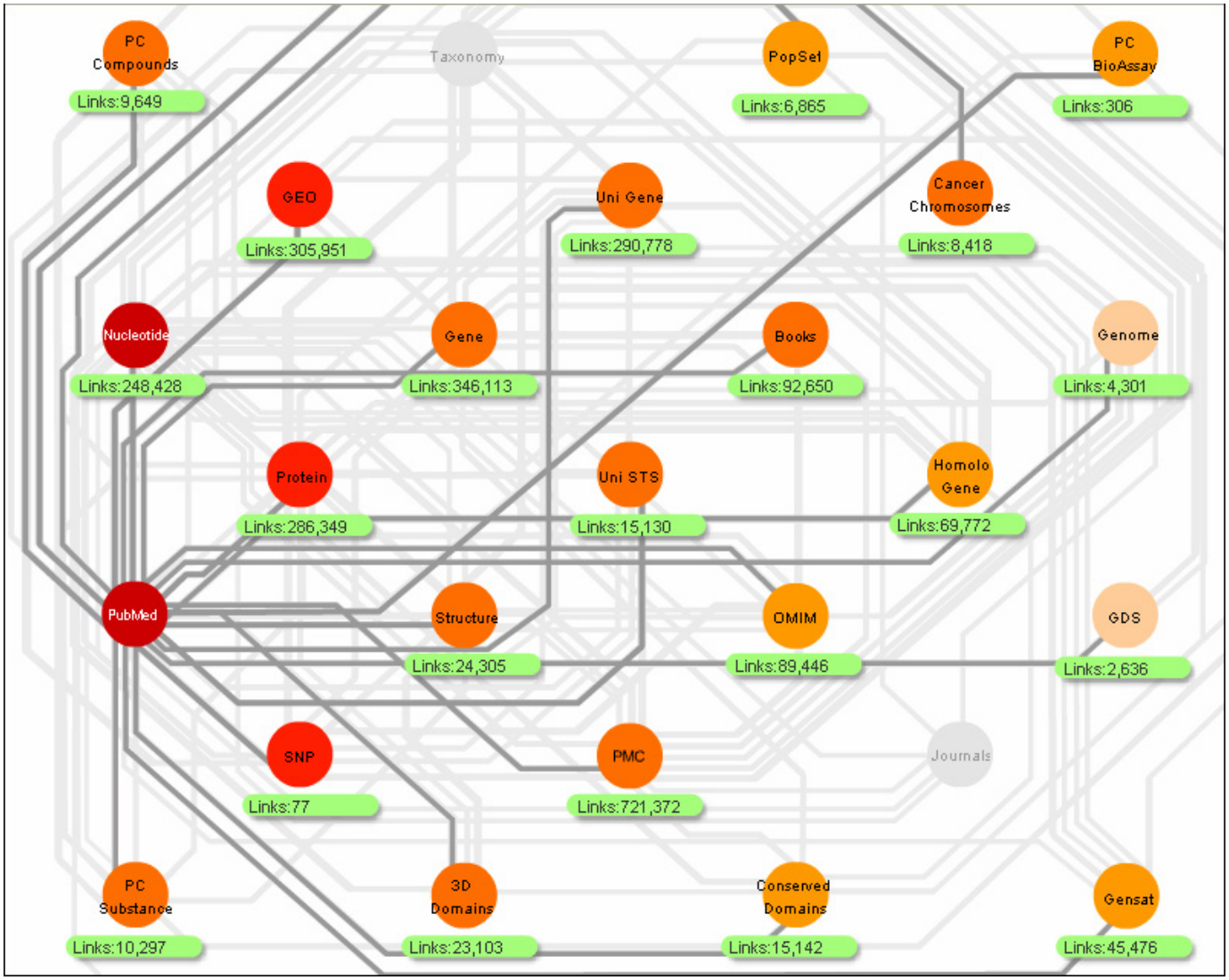
- Dado un UniGene_cluster (tipo Hs.123), obtenemos su GeneID asociado, necesario para realizar el futuro cruce de genes.

Con este *módulo* UniGene hemos obtenido nuestra propia fuente para unificar códigos de gene provinientes de la microarray. Hemos tratado únicamente los datos necesarios con los que trabaja nuestra microarray. El resto de códigos han sido desechados puesto que ocuparían espacio inutil en nuestra base de datos, ya que nunca van a ser empleados.

Tenemos pues la base de conocimiento para que, con un par de consultas, pasemos de un código origen de un tipo determinado a un código final de tipo GeneID, con el que sí podemos realizar operaciones de cruce, consultando su información para todas las bases de datos de *Entrez Gene* diseñadas en el apartado anterior (*Base de datos Gene*).

Nuestro identificador clave de gen es por supuesto el GeneID, y es con él con el que tenemos acceso a todas y cada una de las clases disponibles en nuestra base de datos. A partir del GeneID conocemos toda la información que necesitamos del gen.

5.2.4.- Base de datos PubMed:



Observamos en el gráfico las relaciones existentes entre la base de datos PubMed y el resto de bases de datos.

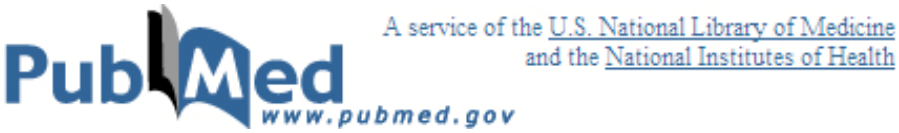
A nosotros nos interesará saber cómo se relaciona con la base de datos Gene, únicamente. La única funcionalidad que nos interesará de la base de datos PubMed será la de conocer en qué publicaciones aparece un gen determinado. O qué genes aparecen en mismas publicaciones que un gen dado.

También será necesario filtrar por título de la publicación. Así pues, puede que nos interese conocer aquellos genes que se encuentran en publicaciones de PubMed que traten el tema de *cáncer de pulmón*.

En la aplicación de cruces, si queremos cruzar datos por PubMed, se incluirá un campo de "concepto" para que el usuario filtre las publicaciones según del contenido que traten.

Además sería interesante conocer aquellos genes que se encuentren (o que se hable de ellos) en las mismas publicaciones que un gen dado. Podemos elegir aquellos genes que tengan misma taxonomía que un gen dado (de la misma especie) o bien todos los genes existentes, sin importar la especie de la que sea.

5.2.4.1.- Identificación de Clases y Atributos:



Toda la información contenida en la base de datos UniGene se encuentra almacenada y disponible para su descarga en el FTP del NCBI:
[ftp://ftp.ncbi.nih.gov/gene]

La estructura jerárquica del directorio FTP nos proporciona los ficheros con información útil para nuestra aplicación:

ftp://ftp.ncbi.nih.gov/
gene/
DATA/
gene2pubmed.gz

En este caso sólo tenemos un fichero con datos relevantes de la relación de Gene con PubMed.
La base de datos de PubMed es mucho más completa, conteniendo toda la información de las publicaciones existentes hasta el momento.

Este fichero sin embargo sólo contiene la lista de publicaciones en las que ha aparecido cada gen de la base de datos de Gene.
Así pues, si deseamos conocer más acerca de dichas publicaciones, necesitaremos hacer una consulta remota a la base de datos PubMed.

Bajarse toda la información de PubMed sería una tarea muy costosa tanto en espacio como en tiempo, ya que los datos que contiene son muy numerosos.

Cómo en nuestra aplicación lo importante es conocer en qué publicaciones aparece un gen, y los genes que aparecen en publicaciones dónde aparece un gen dado, con la información que contiene este fichero podemos construir las consultas necesarias para alcanzar dicha funcionalidad.

Será necesario por tanto hacer uso de las *eUtils* para consultar información específica de un PubMed dado. El fichero gene2pubmed no es más que una relación entre GeneID's y PubMed Id's. Con los PMID's podemos realizar consultas remotas mediante *eUtils* para conseguir, entre otras cosas, ver el nombre de la publicación con ese PMID asignado, o bien para filtrar y obtener aquellas publicaciones relacionadas con *cancer de pulmón* por ejemplo.

Por tanto, la única información que tendremos acerca de PubMed en local serán las relaciones directas entre GeneID's y PMID's.
Con esta simple relación conseguimos los dos requisitos fundamentales de nuestra aplicación en base a PubMed:

- Publicaciones en las que aparece un gen origen.
- Genes que aparecen en mismas publicaciones que un gen origen.

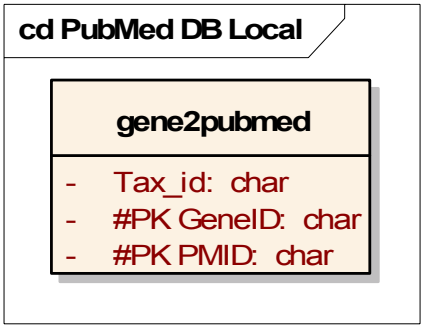
Información contenida en el fichero gene2pubmed

gene2pubmed			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
Tax_id	Texto (10)	V	Identificador para la especie del gen sobre el que trata la publicación.
GeneID	Texto (10)	V	Identificador único de gen.
PMID	Texto (10)	V	Identificador único de la publicación de PubMed

Toda esta información será útil y necesaria para la funcionalidad de nuestra aplicación. La minería de datos en este caso no tiene sentido, ya que vamos a tomar el contenido completo del fichero.

Como vemos, la clave primaria la componen los campos GeneID y PMID, dando a entender que dados un id de gen y un id de pubmed, sólo existe una entrada para la base de datos.

Como hemos dicho antes, para obtener la información relevante asociada a un PMID, cómo puede ser el título de la publicación o si contiene una serie de términos clave, habrá de realizarse una consulta remota a *eUtils* con el PMID adecuado.



Esta será la clase PubMed que tendremos en nuestra base de datos local, junto con sus atributos. Sólo mantenemos la información relevante para realizar nuestras consultas. El resto de información se obtendrá remotamente.

El campo de Tax_id es necesario para el tipo de consulta en el que se quiere conocer las publicaciones de un gen con esa taxonomía, o bien, todas las publicaciones de ese gen para todos las especies en las que aparece.

La información que será accesible de PubMed a través de *eUtils* y dado un PMID base será la siguiente:

PubMed Remoto	
ATRIBUTOS	**DESCRIPCIÓN**
ALL	Todos los términos de todos los campos de búsqueda.
UID	Identificador único de la publicación.
FILT	Límite de entradas.
TITL	Título de la publicación.
WORD	Texto completo asociado a la publicación.
MESH	Medical Subject Headings asignados a la publicación.
MAJR	Términos MesH más importantes de la publicación.
AUTH	Autores de la publicación.
JOUR	Abreviación de la revista dónde aparece la publicación.
AFFL	Instituación a la que se encuentra afiliado el autor y dirección.
ECNO	EC number for enzyme or CAS registry number.
SUBS	CAS chemical name or MEDLINE Substance Name.
PDAT	Fecha de la publicación.
EDAT	Primera fecha a partir de la cual la publicación fue accesible a través de Entrez.
VOL	Número de volumen de la publicación.
PAGE	Número de páginas.
PTYP	Tipo de publicación.
LANG	Lenguaje de la publicación.
ISS	Tema de la publicación.
SUBH	Especificaciones adicionales para el término MESH.
SI	Referencia para acceder a la publicación en otras bases de datos.
MHDA	Fecha de la publicación indexada con términos MESH.
TIAB	Texto del título y del resumen de la publicación.
OTRM	Otros términos asociados a la publicación.
INVR	Investigador.
COLN	Autor corporativo.
CNTY	País.
PAPX	MeSH pharmacological action pre-explosions.
GRNT	NIH Grant Numbers.
MDAT	Fecha de la última modificación.
CDAT	Fecha de finalización de la publicación.
PID	Identificador de la editorial.

FAUT	Primer autor.
FULL	Nombres completos de los autores.
FINV	Nombre completo del investigador.
LAUT	Último autor.
PPDT	Fecha de impresión.
EPDT	Fecha de publicación electrónica.
LID	ID de la localización electrónica.

Todos estos campos son los datos asociados a una publicación con un PMID determinado. Como vemos, son muchos y muy amplios, por lo que hemos tomado la decisión de no descargarlos a la base de datos local.

Esta decisión ha sido tomada en base a varios aspectos:

- No sobrecargar la base de datos local con información no útil para la aplicación de consultas.
- No sobrecargar los robots de descargas bajando ficheros del orden de varios Gigabytes.
- En la aplicación, el acceso a estos campos será muy puntual. Sólo para aquellas consultas que requieran emplear un parámetro de filtrado cómo por ejemplo: *Obtener aquellos genes que aparezcan en publicaciones relacionadas con el cáncer de pulmón.*

En este último caso, tendríamos que hacer una consulta remota empleando *eUtils,* consultando aquellas publicaciones que traten el tema del cáncer de pulmón.

La llamada a *eUtil* sería cómo sigue:

[http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed&term="cancer%20lung[WORD]"&retmax=5000000]

De esta forma, obtenemos aquellos PMID's de publicaciones que contengan el texto Cancer de Pulmón en sus páginas.

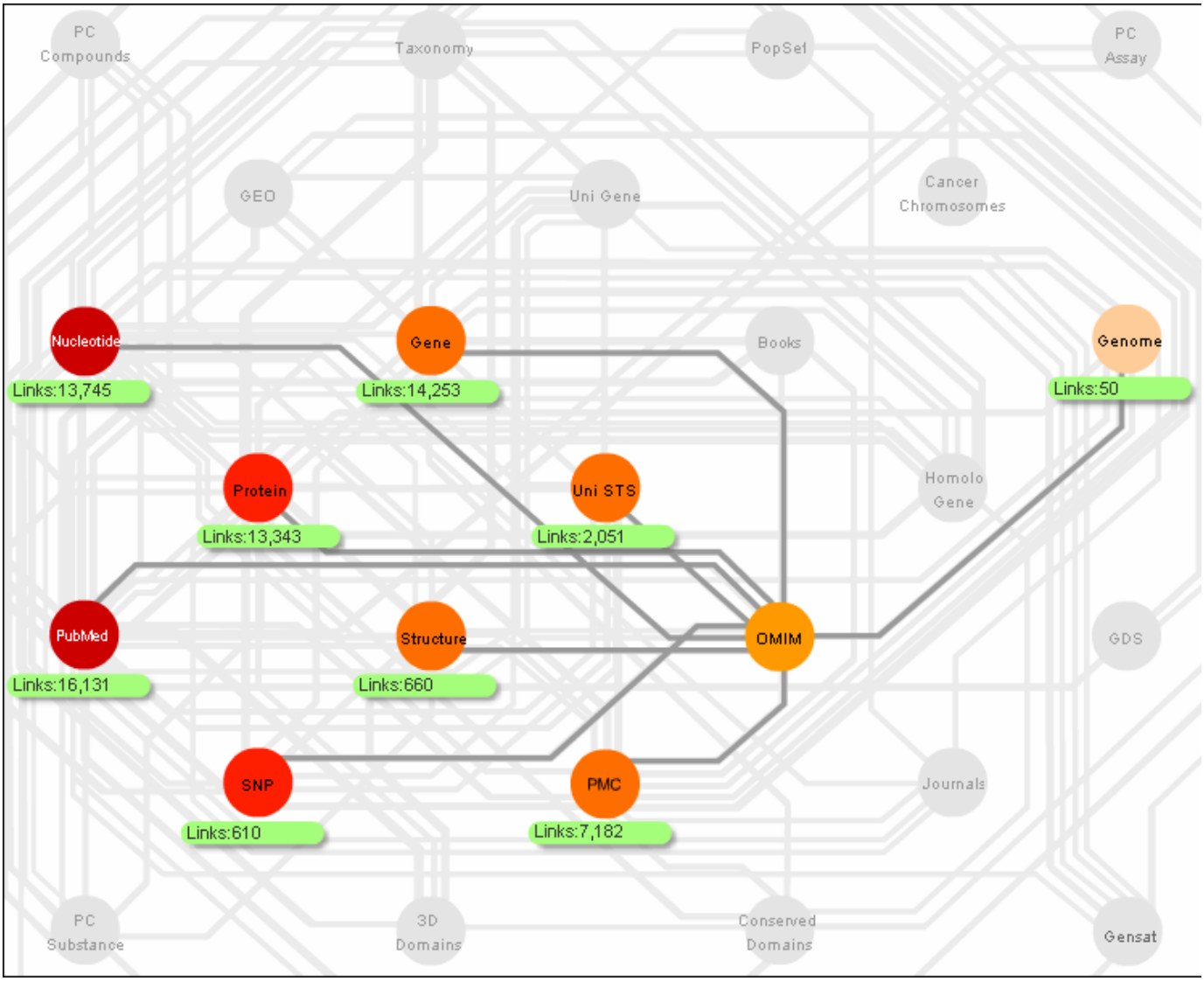
Ya con estos PMID's podríamos trabajar en nuestra base de datos local, ya que tenemos la tabla que los hace corresponder con GeneID's (todos los genes que aparecen en dicha publicación identificada por su PMID). Y cómo hemos visto anteriormente, con GeneID podemos consultar cualquier información relacionada con los genes.

Si hubiéramos querido aquellas publicaciones que contengan la frase cáncer de pulmón en el título de la publicación, la consulta habría sido:

[http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed&term="cancer%20lung[TITL]"&retmax=5000000]

Cómo vemos, sólo hemos cambiado el campo de búsqueda entre corchetes. Si queremos que la búsqueda se realice sobre todos los campos, basta con no incluir los corchetes.

5.2.5.- Base de datos OMIM:



Observamos en el gráfico las relaciones existentes entre la base de datos OMIM y el resto de bases de datos.

A nosotros nos interesará saber cómo se relaciona con la base de datos Gene, únicamente. La única funcionalidad que nos interesará de la base de datos OMIM será la de conocer en qué patologias aparece un gen determinado. O qué genes aparecen en mismas patologías que un gen dado.

Cada patología tiene asociado un código MIM, de manera que nuestra base de datos local hará corresponder a un gen (identificado por su GeneID) aquellas patologías en las que se encuentre mediante una asociación GeneID con código MIM.

Así pues, partiendo de GeneID's podremos cónocer las patologías en las que aparecen y, en el sentido opuesto, dada una patología concreta, podremos conocer todos los genes que intervienen o están presentes de alguna forma en ella.

Estas dos funcionalidades serán las primordiales que llevará a cabo nuestra aplicación de cruces de información.

5.2.5.1.- Identificación de Clases y Atributos:

Toda la información contenida en la base de datos OMIM se encuentra almacenada y disponible para su descarga en el FTP del NCBI:
[ftp://ftp.ncbi.nih.gov/gene]

La estructura jerárquica del directorio FTP nos proporciona los ficheros con información útil para nuestra aplicación:

ftp://ftp.ncbi.nih.gov
gene/
DATA/
mim2gene.gz

En este caso sólo tenemos un fichero con datos relevantes de la relación de Gene con OMIM.
La base de datos de OMIM es mucho más completa, conteniendo toda la información de las patologías existentes hasta el momento.

Este fichero sin embargo sólo contiene la lista de patologías en las que ha aparecido cada gen de la base de datos de Gene.
Así pues, si deseamos conocer más acerca de dichas patologías, necesitaremos hacer una consulta remota a la base de datos OMIM.

Bajarse toda la información de OMIM sería una tarea muy costosa tanto en espacio como en tiempo, ya que los datos que contiene son muy numerosos y su uso en nuestra aplicación se dará en contadas circunstancias.

Cómo en nuestra aplicación lo importante es conocer en qué patologías aparece un gen, y los genes que aparecen en patologías dónde aparece un gen dado, con la información que contiene este fichero podemos construir las consultas necesarias para alcanzar dicha funcionalidad.

Será necesario por tanto hacer uso de las *eUtils* para consultar información específica de un código MIM dado. El fichero mim2gene no es más que una relación entre GeneID's y OMIM Id's. Con los OMIM ID's podemos realizar consultas remotas mediante *eUtils* para conseguir, entre otras cosas, ver el nombre de la patología.

Por tanto, la única información que tendremos acerca de OMIM en local serán las relaciones directas entre GeneID's y MIM ID's.
Con esta simple relación conseguimos los dos requisitos fundamentales de nuestra aplicación en base a OMIM:

- Patologías en las que aparece un gen origen.
- Genes que aparecen en mismas patologías que un gen origen.

Información contenida en el fichero mim2gene (nosotros emplearemos el nombre gene2mim para mantener coherencia con el resto de clases obtenidas hasta el momento).

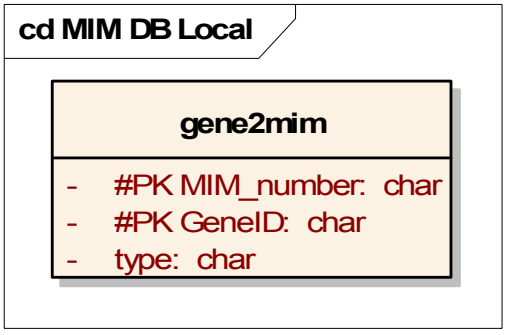
gene2mim			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
MIM_number	Texto (10)	**V**	Identificador único para la patología.
GeneID	Texto (10)	**V**	Identificador único de gen.
type	Texto (10)	**V**	Tipo de la relación entre el gen y la patología. Puede ser "gene" cuando el MIM está asociado a un gen del que no se conoce la patología. O bien puede ser "phenotype" cuando el MIM está asociado a una enfermedad que está asociada con dicho gen.

Toda esta información será útil y necesaria para la funcionalidad de nuestra aplicación. La minería de datos en este caso no tiene sentido, ya que vamos a tomar el contenido completo del fichero.

Como vemos, la clave primaria la componen los campos GeneID y MIM_number, dando a entender que dados un id de gen y un id de OMIM, sólo existe una entrada para la base de datos.

Como hemos dicho antes, para obtener la información relevante asociada a una enfermedad, cómo puede ser el título de la enfermedad, habrá de realizarse una consulta remota a *eUtils* con el MIM_number adecuado.

El type será relevante en el hecho de que, cuando sea de tipo "gene" el resultado podrá ser desechado porque no se conoce la enfermedad. Los únicos resultados útiles serán aquellos cuyo type sea del tipo "Phenotype".



Esta será la clase OMIM que tendremos en nuestra base de datos local, junto con sus atributos. Sólo mantenemos la información relevante para realizar nuestras consultas. El resto de información se obtendrá remotamente.

La información que será accesible de OMIM a través de *eUtils* y dado un MIM_number base será la siguiente:

OMIM Remoto	
ATRIBUTOS	**DESCRIPCIÓN**
ALL	Todos los términos de todos los campos de búsqueda.
UID	Identificador único de la patología.
FILT	Limita el número de entradas.
TITL	Título de la patología.
WORD	Texto completo asociado a la patología.
AUTH	Autor de la entrada OMIM.
CLIN	Características clínicas de la enfermedad.
MDAT	Última fecha de modificación.
ALVR	Conjunto de mutaciones provocadas por la enfermedad.
MDHS	Todas las fechas en que la entrada fue actualizada.
REFR	Autores y citas.
GMAP	Map Location del cromosoma.
DSDR	Texto en el campo de desórdenes.
GENE	Nombre del gen asociado a la entrada OMIM.
ECNO	EC number for enzyme or CAS registry number.
CHR	Número de cromosoma.
EDTR	Editor de la entrada de la patología.
PROP	Propiedades.
PDAT	Fecha de primera aparición de la entrada en la base de datos.

Todos estos campos son los datos asociados a una patología con un MIM_number determinado. Como vemos, son muchos y muy amplios, por lo que hemos tomado la decisión de no descargarlos a la base de datos local.

El único empleo de los datos remotos de OMIM en la aplicación será para mostrar en el resultado final el nombre de las patologías que se han encontrado.

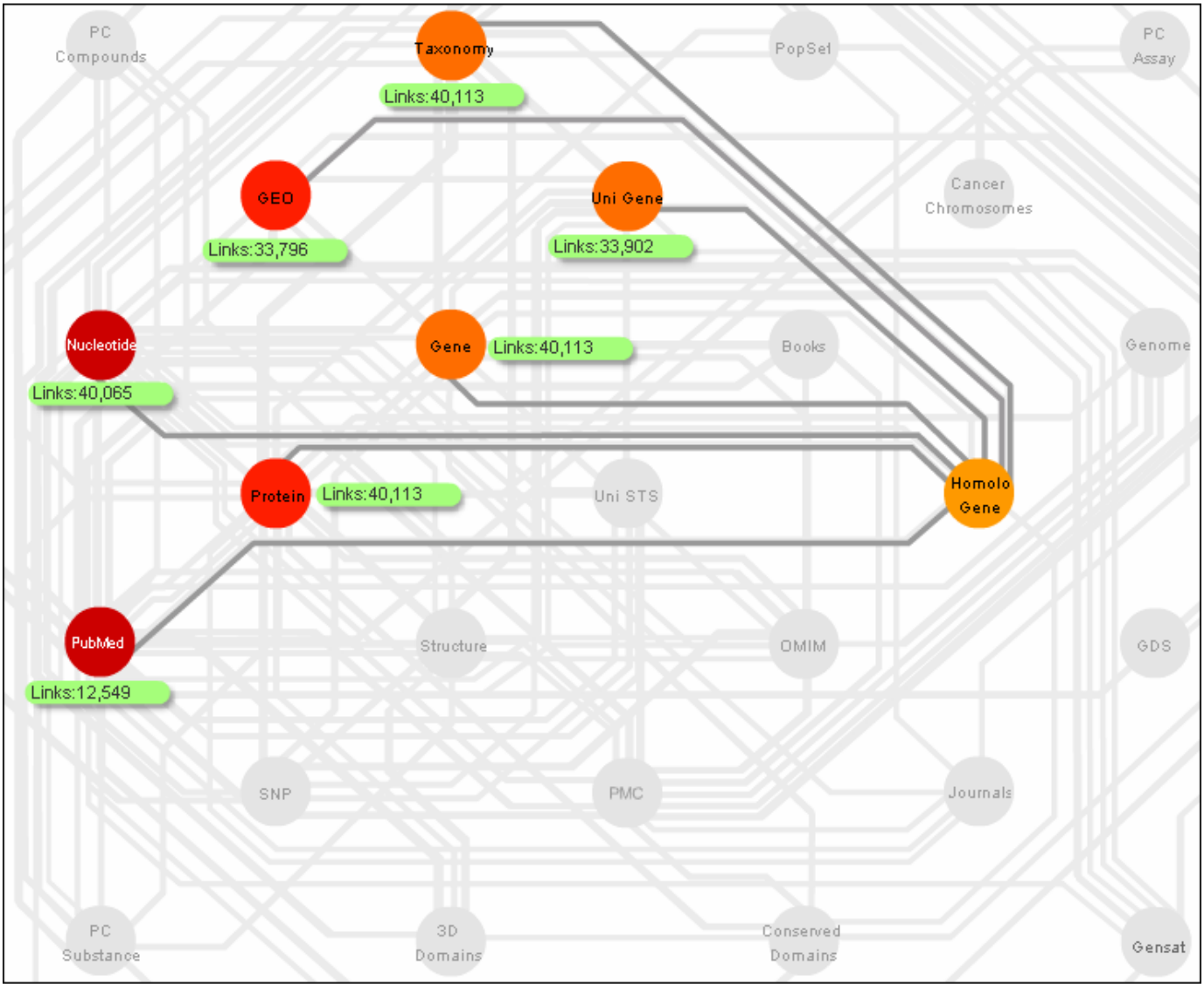
En este último caso, tendríamos que hacer una consulta remota empleando *eUtils,* consultando el nombre de la patología a través de su MIM_number.

La llamada a *eUtil* sería cómo sigue:

[http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=omim&id=MIM_number]

De esta forma, obtenemos el nombre de la patología asociada a dicho MIM_number.
[El resultado deberá ser parseado por el php que muestra los resultados, ya que esummary muestra más información de la que necesitamos]

5.2.6.- Base de datos Homologene:



Observamos en el gráfico las relaciones existentes entre la base de datos Homologene y el resto de bases de datos.

A nosotros nos interesará saber cómo se relaciona con la base de datos Gene, únicamente. La única funcionalidad que nos interesará de la base de datos Homologene será la de conocer todos los genes que sean homólogos a un gen dado, con el objetivo principal de descartarlos en el resultado final.

Nuestra aplicación de cruces on-line, devolverá como resultado una serie de genes que cumplen las condiciones de búsqueda impuestas por el usuario. Estos genes resultados puede que sean homólogos entre ellos y la información que se estré mostrando como final puede estar duplicada.

La principal funcionalidad de emplear la base de datos Homologene es filtrar los resultados, mostrando sólo aquellos genes que son totalmente independientes entre sí, no son homólogos.

5.2.6.1.- Identificación de Clases y Atributos:

El empleo de Homologene no será muy común en la aplicación. Únicamente se empleará como filtro final, una vez que se han obtenido todos los genes resultados de los cruces según los parámetros indicados por el usuario.

Este filtro final consistirá en descartar de los resultados aquellos genes que sean homólogos entre sí.

Puesto que el empleo de Homologene es a voluntad del usuario, se ha decidido por no traer a la base de datos local la información que contiene.
Esta información no será más que una lista de GeneID's que serán homólogos a otros GeneID's.

La consulta de filtro de homólogos se realizará pues remotamente, empleando *eUtils*. Dicha consulta se realizará para cada gen solución obtenido.
Los genes que aparezcan como homólogos se eliminarán de la solución y no se consultará de nuevo por sus homólogos. De esta forma disminuimos el número de consultas remotas ya que, como comentamos en el estudio de viabilidad, la realización de muchas consultas al servidor del NCBI puede cortarnos la conexión durante un periodo de tiempo.

La información que será accesible de Homologene a través de *eUtils* será la siguiente:

Homologene Remoto	
ATRIBUTOS	**DESCRIPCIÓN**
ALL	Todos los términos de todos los campos de búsqueda.
UID	ID único asignado a la publicación.
FILT	Para limitar el número de entradas.
TITL	Título de la homología.
WORD	Texto completo de la referencia de los genes homólogos.
PROP	Propiedades.
ORGN	Nombre científico y común del organismo.
GNID	GeneID
GENE	Nombre del gen.
GDSC	Descripción del gen.
PUID	Identificador de la proteina.
PRAC	Accesos de la proteina.
NUID	Identificador nucleótido de las secuencias.
NCAC	Accesos nucleotidos de las secuencias.
UGID	UniGene ID.

ANCS	Nombres científico y común del organismo antecesor.
DOM	Nombre de dominio.

Todos estos campos son los datos asociados a una homología.

El único empleo de los datos remotos de Homologene en la aplicación será para obtener los genes homólogos a uno dado.

La llamada a *eUtil* sería cómo sigue:

[http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=homologene&term=GeneID[GNID]]

Con esta llamada conseguimos el identificador Homologene asociado al GeneID del gen introducido como origen de la consulta remota.
Con la siguiente llamada, conseguimos los GeneID's de los genes que son homólogos con dicho gen.

Estos genes se obtienen empleando el identificador de Homologene obtenido en la consulta anterior.

[http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=homologene&id=ID_HOMOLOGENE]

El resultado de esta consulta habrá que parsearlo, quedándonos únicamente con aquellos campos que indican los GeneID's pertenecientes a la homología identificada con el identificador ID_HOMOLOGENE. Esta homología es la que presenta el gen origen de la consulta anterior.

5.2.7.- Base de datos KEGG:

La base de datos KEGG es una base de datos japonesa, que almacena conocimiento para comprender información funcional de alto nivel de las células y organismos que forman parte del complejo genómico y molecular.

KEGG consiste en múltiples bases de datos divididas en tres categorías:
- Systems Information.
- Genomic Information.
- Chemical Information.

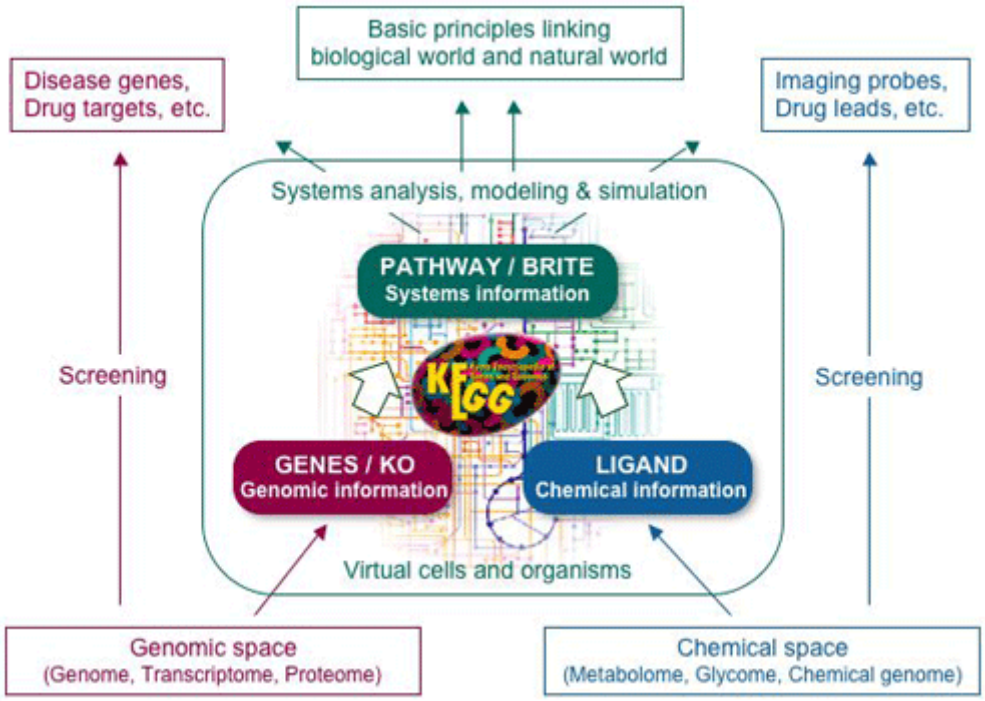
Las bases de datos contenidas en cada categoría son las siguientes:

Bases de Datos del KEGG		
Base de datos	**Contenido**	**Categoría**
KEGG PATHWAY	Mapas de Pathways	Systems Information
KEGG BRITE	Jerarquías funcionales	Systems Information
KEGG MODULE	Módulos de Pathways	Systems Information
KEGG DISEASE	Enfermedades Humanas	Systems Information
KEGG ORTHOLOGY	KO Groups	Genomic Information
KEGG GENES	Genes en genomas de alta calidad	Genomic Information
KEGG DGENES	Genes en genomas en borrador	Genomic Information
KEGG EGENES	Genes as EST contigs	Genomic Information
KEGG GENOME	Organismos con genomas completos	Genomic Information
KEGG SSDB	Secuencias similares y relaciones importantes	Genomic Information
KEGG COMPOUND	Metabolismos	Chemical Information
KEGG DRUG	Drogas	Chemical Information
KEGG GLYCAN	Glycans	Chemical Information
KEGG ENZYME	Enzimas	Chemical Information
KEGG REACTION	Reacciones enzimáticas	Chemical Information
KEGG RPAIR	Reparaciones y transformaciones químicas	Chemical Information

El conjunto de bases de datos de Genomic Information se denominan KEGG LIGAND.

KEGG es una base de datos de sistemas biológico, consistente en bloques genéticos de genes y proteína (*KEGG GENES*), bloques químicos de sustancias endógenas y exógenas (*KEGG LIGAND*), diagramas moleculares de redes de interacción y reacción (*KEGG PATHWAY*), y jerarquías y relaciones de varios objetos biológicos (*KEGG BRITE*).

KEGG proporciona una base de conocimiento como referencia para enlazar genomas con sistemas biológicos y entornos, medianto los procesos de PATHWAY y de BRITE:



La base de datos de PATHWAY's contiene interacciones moleculares y redes de reacción para metabolismos, varios procesos celulares, y enfermedades humanas.

La base de datos de BRITE contiene jerarquías funcionales representando el conocimiento en varios aspectos de sistemas biológicos.

Para nuestra aplicación, la información acerca de la base de datos KEGG será aquella que relacione unos genes con otros en los mismos mapas de PATHWAYS.
Por tanto, el contenido de las diferentes bases de datos japonesas no nos servirán y, únicamente nos fijaremos en la KEGG PATHWAY.

Con la información que contenga esta base de datos deberíamos poder hacer consultas del estilo:

- Obtener mapas de KEGG PATHWAY de un gen determinado.
- Obtener genes que aparezcan en mismo mapa de KEGG PATHWAY que un gen determinado.

5.2.7.1.- Identificación de Clases y Atributos:

Toda la información contenida en la base de datos KEGG PATHWAY se encuentra almacenada y disponible para su descarga en el FTP del KEGG:
[ftp://ftp.genome.jp/pub/kegg/]

La estructura jerárquica del directorio FTP nos proporciona los ficheros con información útil para nuestra aplicación:

ftp://ftp.genome.jp/pub/kegg/
genes/
ko

Trabajaremos con el fichero KO, en el que aparecen todos los Pathways para cada gen existente.
Habremos de recorrerlo y almacenar en la base de datos local la información relevante para usar luego en los cruces.

Esta información será principalmente una lista que enlaza los genes existentes con todos los pathways en los que aparece. Así, podremos conocer en que mapas de pathways aparece y, sobre todo, saber qué genes aparecen en los mismos mapas, para asir conseguir la funcionalidad básica de KEGG para nuestra aplicación.

Existe un problema fundamental, y es que KEGG no emplea GeneID's al no tratarse de una base de datos del NCBI, sino que es totalmente independiente, competencia más bien.

Al no tratar GeneID's no podremos hacer referencia a los datos de genes que hemos obtenido anteriormente para las otras bases de datos. Deberemos idear una forma para obtener el GeneID a partir de los datos que nos proporciona KEGG.

En lugar de GeneID, KEGG asigna a cada gen su propio código KO, pero el nombre del gen sigue siendo el mismo, tanto en NCBI como en KEGG. Con este hecho, podremos conseguir los GeneIDs de los datos que aparezcan en KEGG a través del nombre de gen, ya que en las bases de datos locales de *gene_info* tenemos almacenado un *Description* para cada GeneID.

Así pues, nuestra clase local con información relevante a KEGG no enlazará con el resto de clases mediante el GeneID, sino que enlazará con la clase *gene_info* a través del campo *Description* que determina el nombre del gen.

Es posible que los nombres de gen se diferencien un poco en KEGG y en NCBI, pero se tomará aquel gen cuyo nombre sea más parecido al que aparece en los pathways de KEGG.

En todo caso, si el nombre de gen no aparece en nuestra base de datos local de *gene_info*, o no se encuentra uno parecido, se desechará el resultado por no encontrar el gen correcto y no mostrar información errónea.

Información contenida en el fichero ko

ko			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
ENTRY	Texto (10)	**V**	Identificador KO para el gen.
Name	Texto (255)	**V**	Nombre de la entrada en el pathway.
Definition	Texto (255)	**V**	Nombre del gen.
Class	Lista		Lista de mapas pathways (Nombre y Código) en los que aparece el gen.
Dblinks	Lista		Lista de enlaces a bases de datos externas.
Genes	Lista		Lista de símbolos de genes y taxonomía que están relacionados con el gen actual.

Toda esta información no es necesaria para nuestra aplicación.

Para poder realizar los cruces de información relevantes a KEGG, necesitaremos el nombre del gen, así cómo en el pathway que aparece y el nombre de dicho pathway.

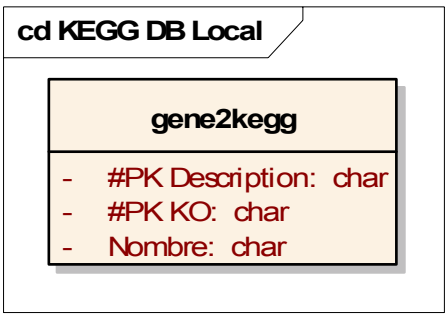
Por tanto, de los datos mostrados arriba, deberemos hacer un tratamiento, de manera que obtengamos del fichero el nombre del gen (*Definition*) y una lista de mapas de pathways en los que aparece:

gene2kegg			
ATRIBUTOS	**DOMINIO**	**NO NULO**	**DESCRIPCIÓN**
Description	Texto (255)	**V**	Nombre del gen.
KO	Texto (10)	**V**	Identificador del mapa de pathway (formato [ko00001]).
Nombre	Texto (255)		Nombre del mapa de pathway.

Nuestra clase local se denominará *gene2kegg* para estar en consonancia con el resto de clases del dominio.

Description será el campo Definition que aparece en el fichero ko.
Para obtener el identificador de mapa de pathway y su nombre, habrá que hacer un tratamiento del campo Class, de manera que se fraccione su información para generar nuestra tabla.

Esta labor de parsear el fichero para obtener la información que nos interese será labor del robot que se encargue de descargar el fichero del FTP del KEGG y de introducir su contenido en la base de datos.



Con estos datos finalmente podremos asegurar la funcionalidad de la aplicación en cuanto a la opción de consultas al KEGG:

- Obtener mapas de pathways en los que aparece un gen:
  - A partir del GeneID, habrá que obtener su nombre de gen consultando en la tabla *gene_info*.
  - Una vez conseguido el nombre de gen, consultar en *gene2kegg* aquellas entradas KO y Nombre para el Description (nombre del gen).
  - Cada una de las salidas KO obtenidas será un mapa de pathways en los que aparece el gen origen.
- Obtener genes que aparecen en mismos mapas que un gen dado:
  - A partir del GeneID, habrá que obtener su nombre de gen consultando en la tabla *gene_info*.
  - Una vez conseguido el nombre de gen, consultar en *gene2kegg* aquellas entradas KO y Nombre para el Description (nombre del gen).
  - Para cada una de las salidas KO obtenidas, consultar en *gene2kegg* todos los *Descriptions* que tengan el mismo KO (gene que aparecerán en mismo map).
  - De todos los *Descriptions* obtenidos, generar su GeneID a través de la consulta en la tabla *gene_info*.

Vemos que en el caso del KEGG, la obtención de resultados es más compleja que en el caso de las otras bases de datos debido a que el concepto de GeneID no existe en KEGG.
Es por esto que necesitamos hacer uso del nombre de gen como paso intermedio para obtener el GeneID.

Las consultas a KEGG pueden ser un poco más lentas debido a este motivo, requieren más procesamiento por parte de la aplicación de consultas on-line.

5.2.8.- Diseño Lógico de la Base de Datos Local:

Una vez hemos analizado todas las bases de datos externas que nos harán falta, hemos decidido qué datos utilizar de cada una de ellas y hemos definido el diseño lógico de cada una de ellas por separado, es hora de unificarlas en la base de datos final que emplearemos en la aplicación.

Esta base de datos estará centralizada en el servidor del IBB, en la máquina *revolutionresearch.uab.es.*

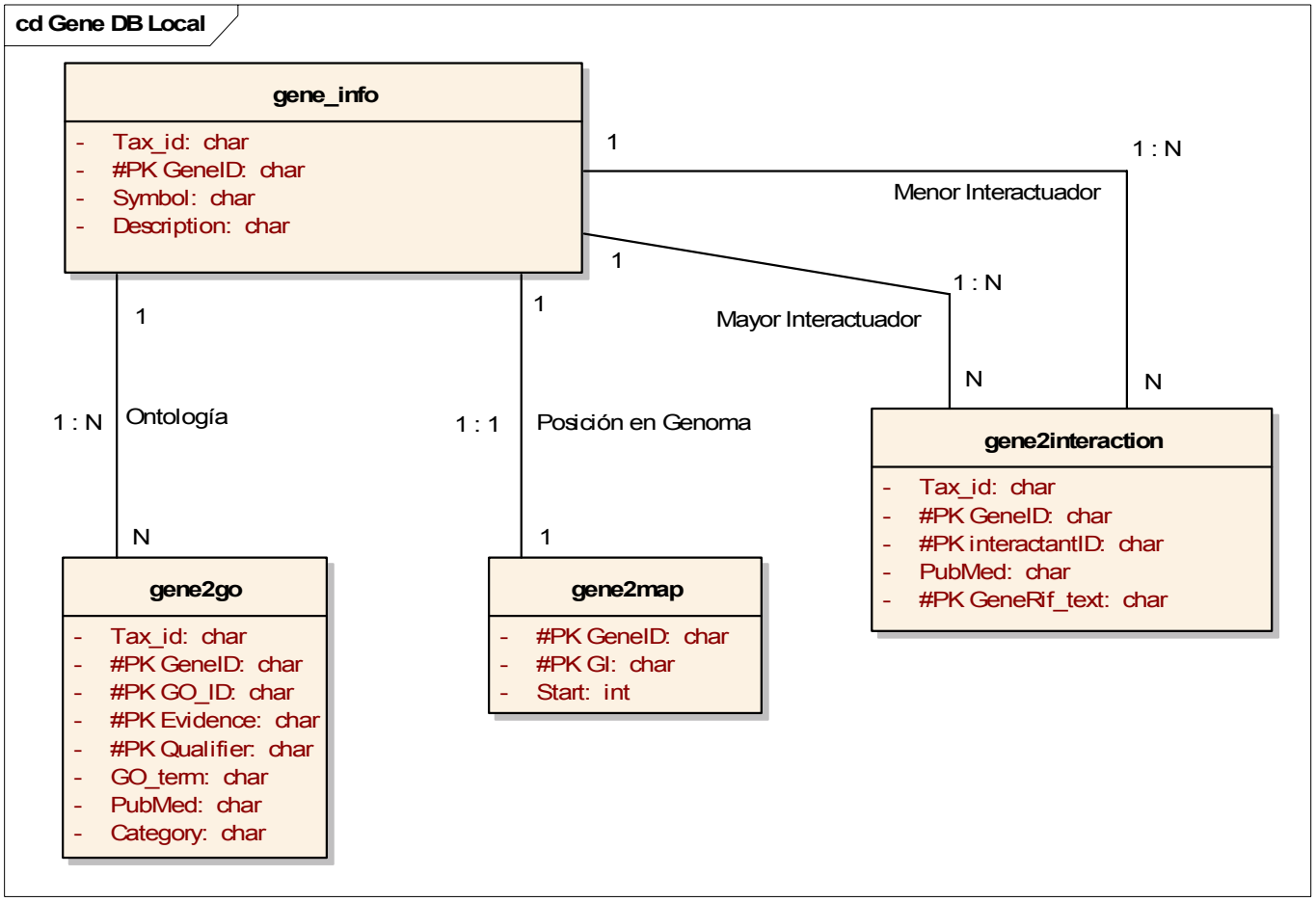
Será actualizada periódicamente por los robots de actualización de datos, teniendo información siempre coherente y no duplicada, semejante a la información que contienen las bases de datos del NCBI.

El diseño de esta base de datos estará centrada en la información de cada gen y a su vez relacionada con todas las características de dicho gen (pubmeds en los que aparece, ontología, patologías, interacciones que presenta, genes vecinos, mapas del KEGG).

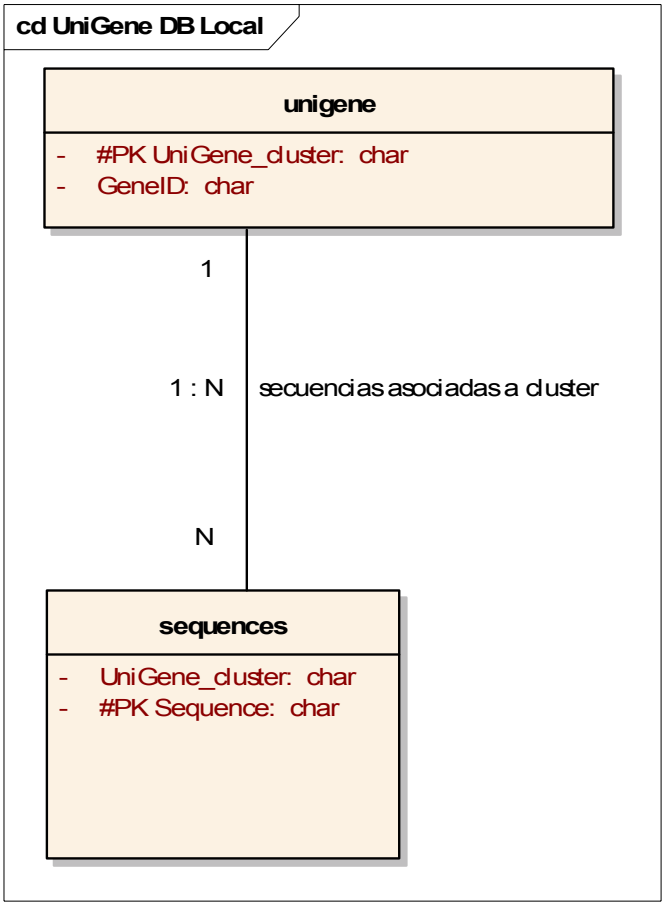
5.2.8.1.- Relaciones entre todas las bases de datos:

Veamos el diseño de cada una de las bases de datos que emplearemos en la aplicación, y cómo se relacionan entre ellas:

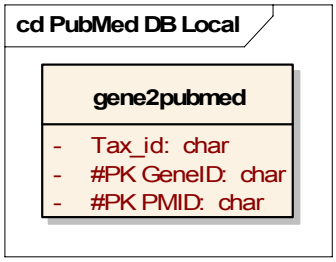
Gene:



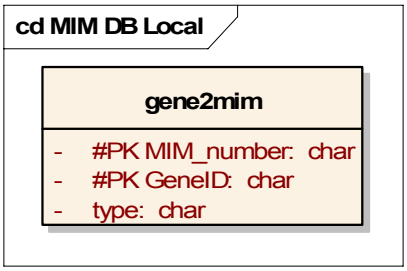
UniGene:



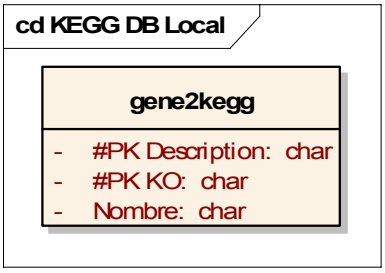
PubMed:



OMIM:



KEGG:



PERTENECE A CLUSTER			
Grado	**Cardinalidad**	**ENTIDADES**	**DESCRIPCIÓN**
2	1:1	**gene_info, unigene**	Un gen pertenece a un cluster de UniGene y cada cluster de UniGene tiene asignado un gen. Con esta relación se obtiene el GeneID asociado a un código de tipo sequencia o UniGene_cluster contenido en la microarray.
La clave primaria de *gene_info* (**GeneID**), hará de clave foránea en *unigene* (**GeneID**) de manera que enlazamos un gen con su cluster de UniGene asociado. Así pues, podemos conocer en qué secuencias aparece un GeneID determinado, o en que cluster de UniGene se encuentra asignado.			

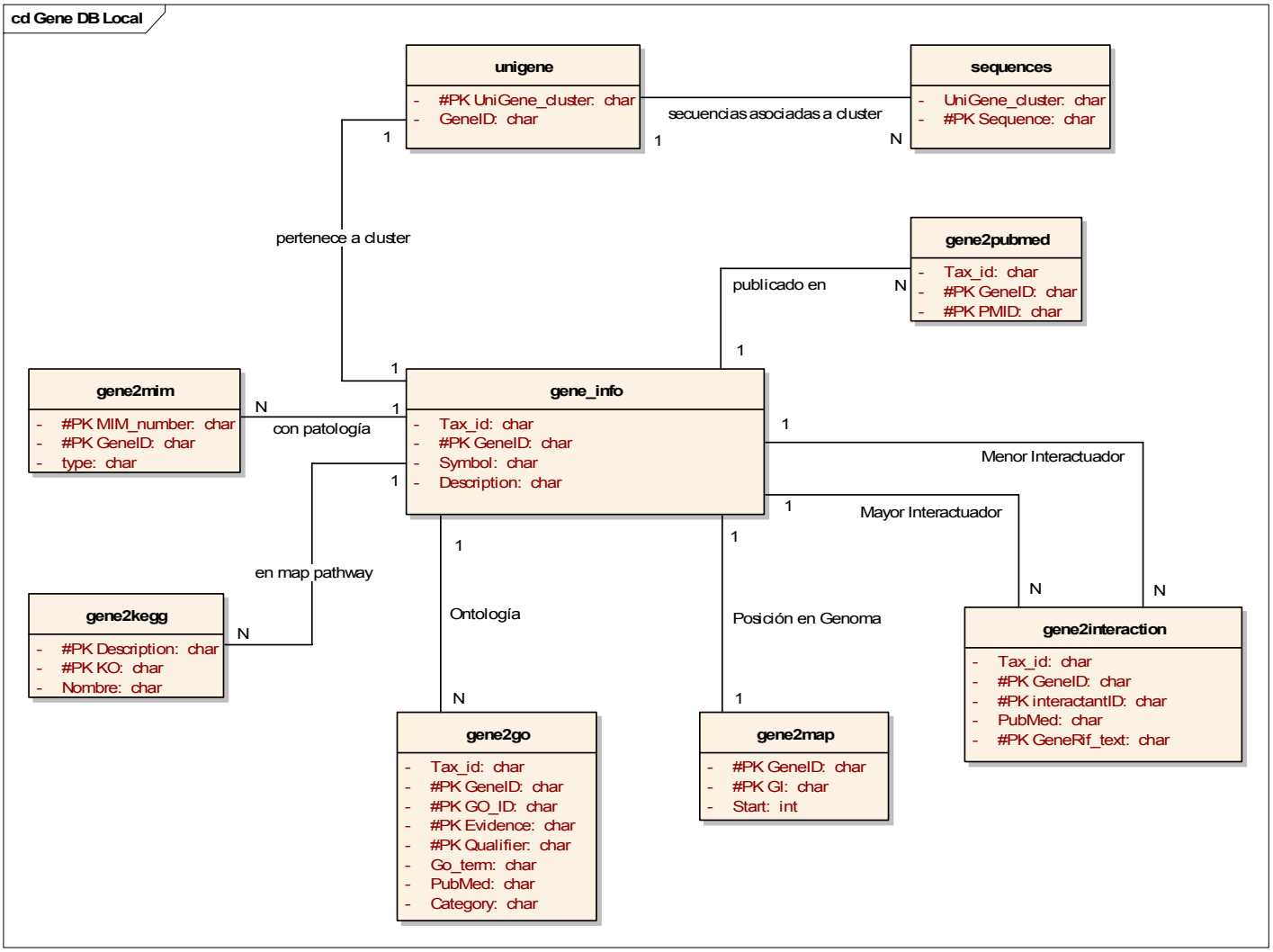
PUBLICADO EN			
Grado	**Cardinalidad**	**ENTIDADES**	**DESCRIPCIÓN**
2	1:N	**gene_info, gene2pubmed**	Un gen puede aparecer en un sin fin de publicaciones del PubMed. Con esta relación se consigue obtener todas las publicaciones en las que ha aparecido un gen determinado.
La clave primaria de *gene_info* (**GeneID**), hará de clave foránea en *gene2pubmed* (**GeneID**) de manera que enlazamos un gen con las publicaciones en las que aparece. Para conocer qué genes aparecen en mismas publicaciones que un gen dado, basta conocer primero las publicaciones en las que aparece éste, para después hacer una consulta sobre pubmed y obtener los GeneID's que también están en esas publicaciones.			

CON PATOLOGÍA			
Grado	**Cardinalidad**	**ENTIDADES**	**DESCRIPCIÓN**
2	1:N	**gene_info, gene2mim**	Un gen tiene asociada una serie de patologías en las que se puede encontrar. Con esta relación se consiguen aquellas patologías en las que interviene un determinado gen.
La clave primaria de *gene_info* (**GeneID**), hará de clave foránea en *gene2mim* (**GeneID**) de manera que enlazamos un gen con las patologías en las que aparece. Para conocer qué genes aparecen en mismas patologías que un gen dado, basta conocer primero las patologías en las que aparece éste, para después hacer una consulta sobre mim y obtener los GeneID's que también están en esas patologías.			

EN MAP PATHWAY			
Grado	**Cardinalidad**	**ENTIDADES**	**DESCRIPCIÓN**
2	1:N	**gene_info, gene2kegg**	Un gen puede aparecer en un determinado número de mapas del KEGG, identificando sus interacciones y reacciones. Con esta relación se consigue conocer aquellos pathways en los que está presente el gen.
En este caso, el campo Description de *gene_info*, hará de clave foránea en *gene2kegg* (**Description**) de manera que enlazamos un gen con los mapas del KEGG en los que se encuentra involucrado. Para conocer qué genes aparecen en mismas mapas que un gen dado, basta conocer primero los mapas en los que aparece éste, para después hacer una consulta sobre kegg y obtener los nombres de genes que también están en esos mapas de interacción.			

Cómo vemos, hemos interrelacionado nuestra base principal de conocimiento (*gene*) con el resto de base de datos de las que será necesario obtener información.

La estructura lógica final de la base de datos local queda como sigue:

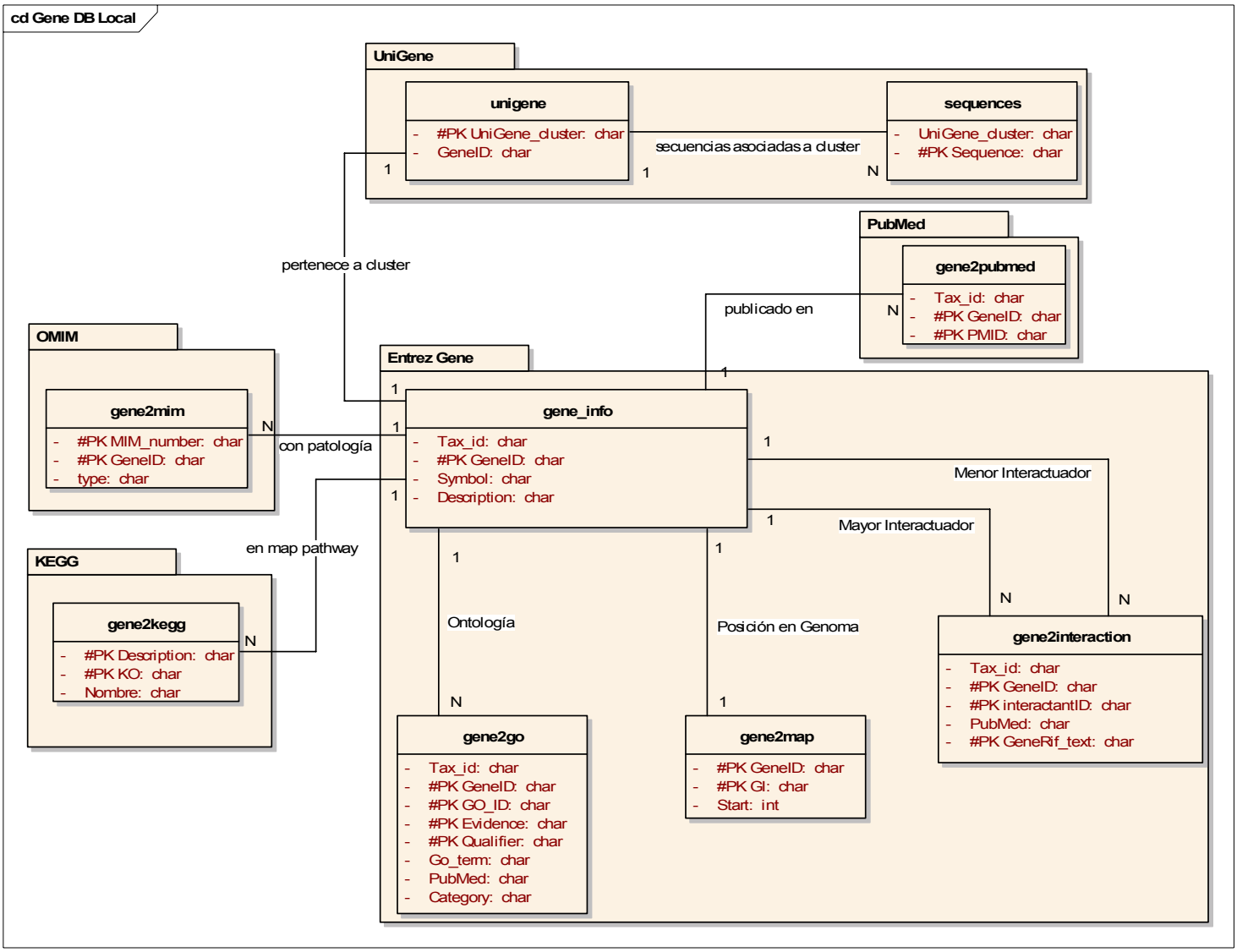


El centro de información, que contiene datos básicos sobre un GeneID, es la tabla gene_info.

A partir de ella, salen relaciones con el resto de tablas estudiadas anteriormente, de manera que se puede generar toda la funcionalidad requerida para el desarrollo de la aplicación:

- Publicaciones en las que aparece un gen y genes que aparecen en misma publicación (*gene2pubmed*).
- Genes con los que interacciona un gen (*gene2interaction*).
- Genes vecinos a un gen (*gene2map*).
- Ontología de un gen y genes con misma ontología (*gene2go*).
- Mapas del KEGG en los que aparece un gen y genes que aparecen en mismos mapas (*gene2kegg*).
- Patologías propias de un gen y genes con mismas patologías (*gene2mim*).
- UniGene cluster asociado a un gen (*unigene*). Esta información es necesaria para el tratamiento de nombres de los genes de la microarray.

En el siguiente gráfico mostramos el diseño lógico de la base de datos local que tendremos en el servidor, encapsulada según las bases de datos remotas de las que hemos obtenido el contenido:



Se observa mejor en este gráfico cómo se comunican entre sí las diferentes bases de datos remotas, empleando cómo código común entre todas ellas el GeneID, a excepción de la base de datos del KEGG que emplea el nombre del gen.

Como veremos posteriormente en la etapa de diseño, todos los datos necesarios serán descargados por los robots de actualización, cada uno de ellos encargado de descargar unos datos y actualizarlos independientemente. Las descargas se realizarán de forma remota, desde aquellos FTP's que hemos visto en el análisis de cada una de las bases de datos externas que necesitaremos.

5.2.8.2.- Integración de la base de datos en el servidor local. Diseño físico:

En el servidor dónde se alojará la base de datos local, ya existe una base de datos en funcionamiento.

Dicha base de datos trata una serie de aplicaciones del departamento en cuanto a tratamiento de microarrays, creación, experimentos y genes que contienen.

Puesto que nuestra aplicación se fundirá posteriormente con la que ya se encuentra realizada, la base de datos que hemos definido anteriormente irá integrada con la ya existente.

El nombre de la base de datos que existe en el servidor local dónde se aloja la aplicación es ***revresearch*** y su estructura física es la siguiente:

applicants							265	MyISAM	utf8_spanish_ci	21.8 KB
downloads							1,706	MyISAM	utf8_spanish_ci	67.6 KB
experiments							534	MyISAM	utf8_spanish_ci	92.8 KB
experiments14							18	MyISAM	utf8_spanish_ci	9.3 KB
experiments15							16	MyISAM	utf8_spanish_ci	7.3 KB
experiments17							133	MyISAM	utf8_spanish_ci	52.8 KB
experiments20							64	MyISAM	utf8_spanish_ci	30.1 KB
gene2go							917,966	MyISAM	utf8_spanish_ci	115.3 MB
gene2interaction							167,916	MyISAM	utf8_spanish_ci	22.9 MB
gene2kegg							9,156	MyISAM	utf8_spanish_ci	972.1 KB
gene2map							3,912,734	MyISAM	utf8_spanish_ci	271.4 MB
gene2mim							16,130	MyISAM	utf8_spanish_ci	885.4 KB
gene2pubmed							3,947,098	MyISAM	utf8_spanish_ci	288.0 MB
gene_ftp							5	MyISAM	utf8_spanish_ci	2.2 KB
gene_info							3,553,603	MyISAM	utf8_spanish_ci	309.1 MB
genes							9,265	MyISAM	utf8_spanish_ci	942.7 KB
lista							0	MyISAM	latin1_general_ci	22.6 KB
microarrays							4	MyISAM	utf8_spanish_ci	2.5 KB
personalize_texp							128	MyISAM	utf8_spanish_ci	8.8 KB
sections							7	MyISAM	utf8_spanish_ci	2.3 KB
sequences							32,432,404	MyISAM	utf8_spanish_ci	1.0 GB
sources							23	MyISAM	utf8_spanish_ci	3.1 KB
unigene							1,620,076	MyISAM	utf8_spanish_ci	62.1 MB
unigene_ftp							176	MyISAM	utf8_spanish_ci	11.8 KB
view							34	MyISAM	utf8_spanish_ci	5.5 KB
25 tabla(s)				Número de filas			46,589,461	--	utf8_spanish_ci	2.1 GB

Estructura de tablas de la base de datos local revresearch.

Hemos creado en la base de datos local las clases definidas previamente, y ya contienen información. Esta información ha sido generada por los robots de actualización remotos cuya función será la de conectarse vía FTP a las bases de datos requeridas, bajar la información y cargarla en la base de datos local.

Observamos que tablas cómo por ejemplo *sequences* tienen más de 32 millones de entradas, y un tamaño de 1Gigabyte.

Cómo hemos dicho anteriormente, los datos a descargar son muy numerosos y de gran tamaño, y por eso la principal razón de haber analizado minuciosamente todas y cada una de las bases de datos existentes es tener en local los mínimos datos que nos proporcionen funcionalidad a la aplicación.

El resto de datos remotos que no nos séan útiles no serán insertados en la base de datos local.

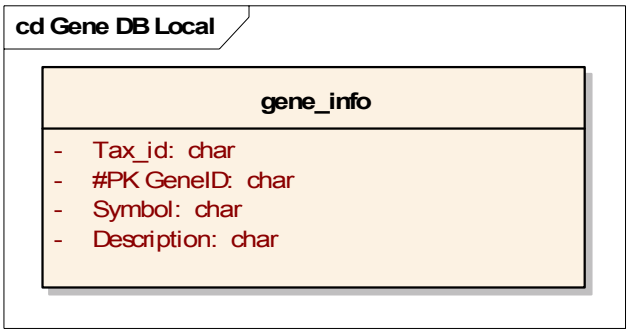
Las tablas correspondientes a nuestra aplicación y ya explicadas en los apartados anteriores son:

- gene2go → contiene la información relativa a las ontologías de cada gen.
- gene2interaction → contiene las interacciones de cada gen.
- gene2kegg → mapas de los pathways del kegg para cada gen.
- gene2map → genes vecinos de cada gen.
- gene2mim → patologías de cada gen.
- gene2pubmed → publicaciones de cada gen.
- gene_info → información básica de cada gen.
- unigene → asigna un UniGene cluster a cada GeneID
- sequences → relación entre los códigos de secuencia y los UniGene clusters.

Veamos cómo la estructura física de cada tabla corresponde con la estructura lógica de cada clase analizada con anterioridad en cada base de datos:

gene_info

Estructura Lógica



Estructura Física

	Campo	Tipo
☐	Tax_id	varchar(10)
☐	GeneID	varchar(10)
☐	Symbol	varchar(20)
☐	Description	varchar(255)

Índices

Índices:

Nombre de la clave	Tipo	Cardinalidad	Acción	Campo
PRIMARY	PRIMARY	3553603		GeneID
Symbol	INDEX	3553603		Symbol
Description	INDEX	710720		Description

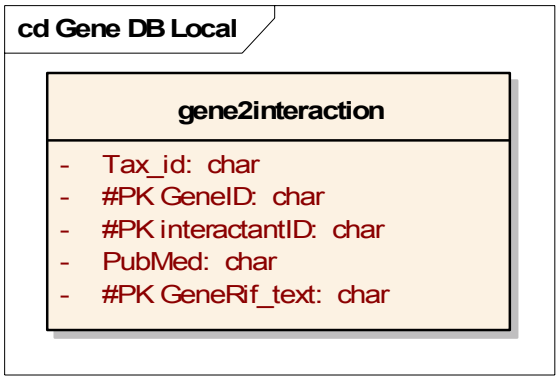
Tenemos un índice en la clave primaria, para agilizar todas las búsquedas para obtener los datos de un determinado *GeneID*.

Además, incluímos un índice en el campo *Symbol*, ya que en ocasiones necesitaremos obtener el GeneID a partir de una combinación entre Symbol y Tax_id.

El índice en el campo *Description* no es más que para obtener los GeneID's cuando se busca por KEGG, ya que KEGG no usa GeneID.

gene2interaction

Estructura Lógica



Estructura Física

Campo	Tipo
Tax_id	varchar(10)
GeneID	varchar(10)
interactantID	varchar(10)
PubMed	varchar(62)
GeneRif_text	varchar(255)

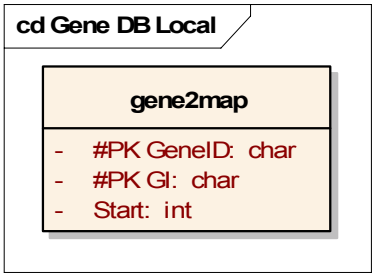
Índices

Nombre de la clave	Tipo	Cardinalidad	Acción		Campo
PRIMARY	PRIMARY	167916			GeneID
					interactantID
					GeneRif_text
GeneID	INDEX	Ninguna			GeneID
interactantID	INDEX	Ninguna			interactantID

Aparte de la clave primaria, tenemos índice en *GeneID* para consultar todos los genes que interactúan con un GeneID (cuando su GeneID es menor que el de los otros genes) y otro en *interactantID* (cuando el GeneID es mayor que el de los otros interactuadores).

gene2map

Estructura Lógica



Estructura Física

Campo	Tipo
GeneID	varchar(10)
GI	varchar(10)
Start	int(10)

Índices

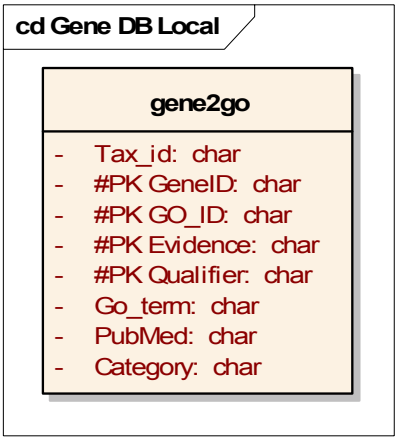
Nombre de la clave	Tipo	Cardinalidad	Acción		Campo
PRIMARY	PRIMARY	3912734			GeneID
					GI
Start	INDEX	3912734			Start
GI	INDEX	130424			GI

Aparte de la clave primaria, tenemos índice en *GI* para centrarnos en aquellos genes que estén en el mismo cromosoma que un gen dado.

Además tenemos índice en el campo *Start* para seleccionar todos aquellos genes que comiencen por posición cercana a la del gen dado dentro del mismo cromosoma.

gene2go

Estructura Lógica



Estructura Física

Campo	Tipo
Tax_id	varchar(10)
GeneID	varchar(10)
GO_ID	varchar(11)
Evidence	varchar(3)
Qualifier	varchar(3)
GO_term	varchar(100)
PubMed	varchar(62)
Category	varchar(10)

Índices

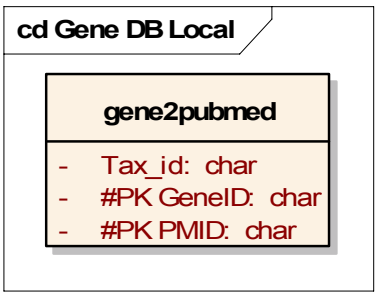
Nombre de la clave	Tipo	Cardinalidad	Acción	Campo
PRIMARY	PRIMARY	917966		GeneID
				GO_ID
				Evidence
				Qualifier
GeneID	INDEX	Ninguna		GeneID
GO_ID	INDEX	Ninguna		GO_ID

Aparte de la clave primaria, tenemos índice en *GeneID* para seleccionar toda la información de GO de un gen determinado.

Además tenemos índice en el campo *GO_ID* para seleccionar todos aquellos genes que presenten mismo comportamiento GO que un gen dado, del cual ya hemos obtenido sus GO_ID's mediante una consulta previa.

gene2pubmed

Estructura Lógica



Estructura Física

Campo	Tipo
Tax_id	varchar(10)
GeneID	varchar(10)
PMID	varchar(10)

Índices

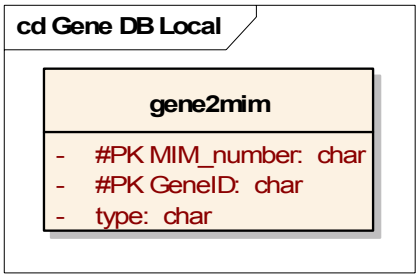
Nombre de la clave	Tipo	Cardinalidad	Acción	Campo
PRIMARY	PRIMARY	3947098		GeneID
				PMID
GeneID	INDEX	Ninguna		GeneID
PMID	INDEX	Ninguna		PMID

Aparte de la clave primaria, tenemos índice en *GeneID* para obtener pubmeds de un gen dado.

Y en *PMID* para seleccionar aquellos genes que aparezcan en mismas publicaciones PMID de un gen que se ha consultado previamente.

gene2mim

Estructura Lógica



Estructura Física

Campo	Tipo
MIM_number	varchar(10)
GeneID	varchar(10)
type	varchar(10)

Índices

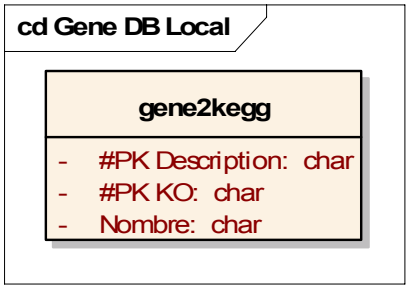
Nombre de la clave	Tipo	Cardinalidad	Acción	Campo
PRIMARY	PRIMARY	16130		MIM_number
				GeneID
MIM_number	INDEX	Ninguna		MIM_number
GeneID	INDEX	Ninguna		GeneID

Aparte de la clave primaria, tenemos índice en *GeneID* para obtener las patologías de un gen dado.

Y en *MIM_number* para seleccionar aquellos genes que presentan mismas patologías que un gen que se ha consultado previamente.

gene2kegg

Estructura Lógica



Estructura Física

Campo	Tipo
Description	varchar(255)
KO	varchar(15)
Nombre	varchar(255)

Índices

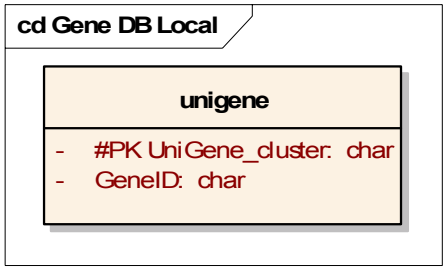
Nombre de la clave	Tipo	Cardinalidad	Acción	Campo
PRIMARY	PRIMARY	9156		Description
				KO

En este caso tenemos como índice la clave primaria.

En la base de datos KEGG buscaremos los KO de un gen a partir de su *Description*, para luego obtener los genes que comparten mapa de pathway del KEGG con él mediante una nueva búsqueda a partir de los *KO's* obtenidos en la consulta previa.

unigene

Estructura Lógica



Estructura Física

Campo	Tipo
UniGene_cluster	varchar(15)
GeneID	varchar(15)

Índices

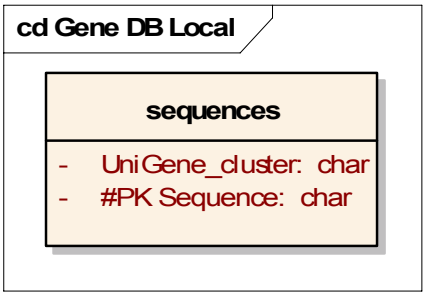
Nombre de la clave	Tipo	Cardinalidad	Acción	Campo
PRIMARY	PRIMARY	1620076		UniGene_cluster
GeneID	INDEX	324015		GeneID

Aparte de la clave primaria, tenemos índice en *GeneID* para obtener el UniGene cluster de un gen dado.

El sentido opuesto, obtener el GeneID de un UnigeneCluster ya viene dado por la clave primaria.

sequences

Estructura Lógica



Estructura Física

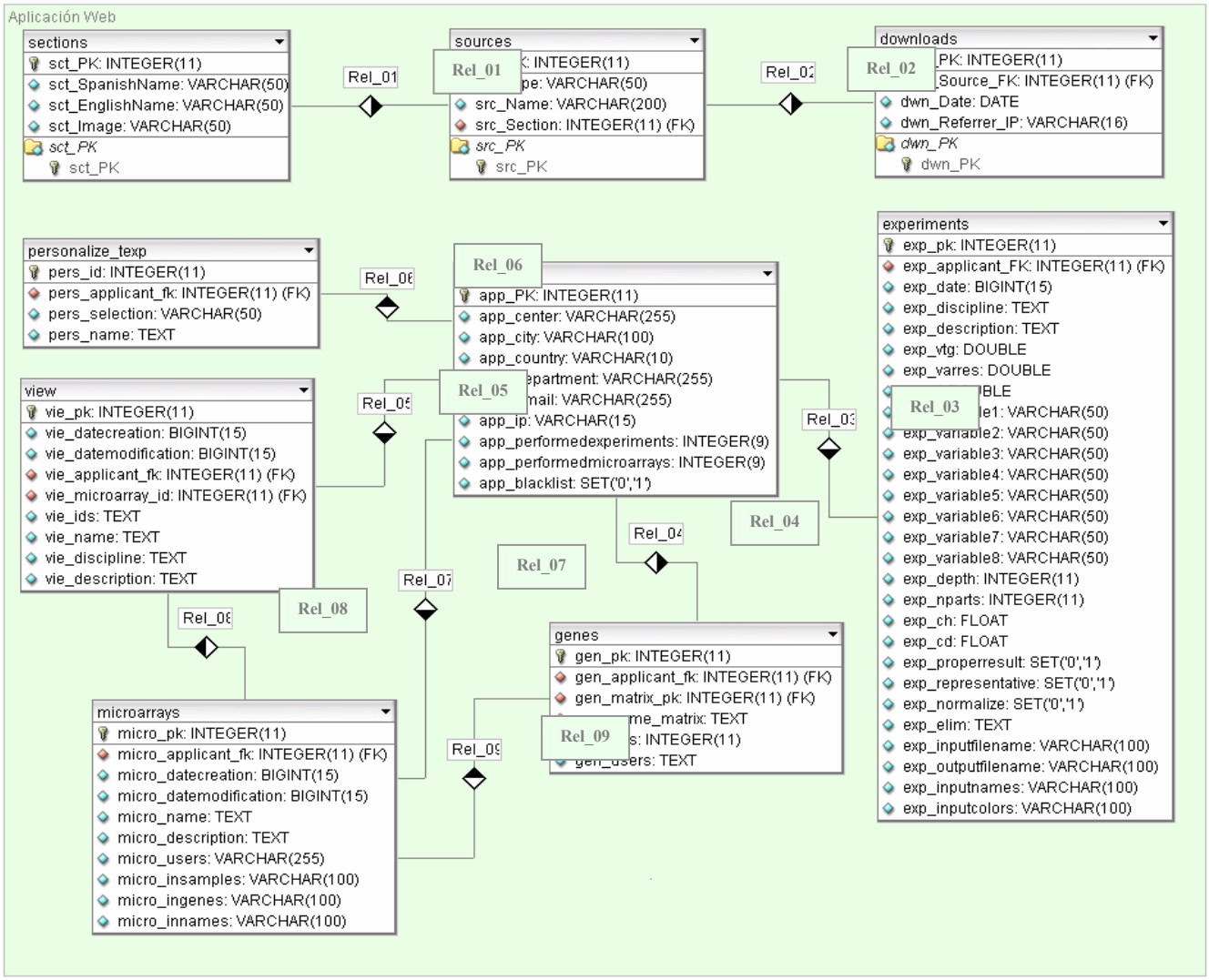
Campo	Tipo
UniGene_cluster	varchar(15)
Sequence	varchar(15)

Índices

Nombre de la clave	Tipo	Cardinalidad	Acción	Campo
PRIMARY	PRIMARY	32432404		Sequence

En este caso, únicamente tendremos de índice la clave primaria.
La funcionalidad de la tabla sequences es obtener el UniGene cluster de un código de secuencia no unificado.
Por esta razón, se buscará el Unigene Cluster de una *sequence* dada como entrada al preproceso de generación de nombres en la microarray.

Una vez hemos analizado la estructura física de las tablas que hemos creado en la base de datos para poder llevar a cabo la funcionalidad completa de nuestra aplicación de cruce de información de genes, analizaremos brevemente el uso que se les da al resto de tablas y su funcionalidad en la aplicación final.



Existen algunas tablas que son secundarias y que no están relacionadas con la aplicación directamente como por ejemplo, *Sections*, *Sources* y *Downloads*. Las tablas *Sections* y *Sources* están vinculadas entre ellas mediante una relación de 1-n, mientras que las tablas *Sources* y *Downloads* están vinculadas mediante una relación de 1-n. La tabla *Sections* se ocupa de guardar los diferentes tipos de secciones que tiene la página web y *Sources*, los diferentes tipos de elementos que contienen cada una de las anteriores. La tabla *Downloads* mantiene una lista de las descargas que se realizan en alguna de las subsecciones y del momento en las que fueron realizadas.

Las principales tablas que aparecen en el modelo son *Applicants*, *Experiments*, *View*, *Microarrays, Genes* y *Personalize_texp*. Veamos con detenimiento, en que consiste cada una de ellas:

- *Applicants* y *Experiments*: la primera tabla se encargará de guardar la lista de usuarios, con todos sus datos, que utilizarán la aplicación. Esta tabla está vinculada con *Experiments*, que se ocupará de almacenar todos los datos de los experimentos que realice cualquier usuario, mediante una relación 1-n ya que un usuario puede tener n experimentos, pero un experimento solo puede pertenecer a un determinado usuario.
- *Personalize_texp*: se ocupa de guardar las preferencias de variables que tiene cada usuario a la hora de visualizar las tablas de experimentos en la aplicación, con lo cual está vinculada con la tabla *Applicants* con una relación de n-1, ya que una preferencia solo puede pertenecer a un usuario, pero en el caso contrario, un usuario puede tener n preferencias. Cabe decir que las preferencias que escoja un usuario, tan solo serán válidas durante la sesión de trabajo en las que fueron escogidas; en el instante que inicie una nueva sesión, el usuario verá en las tablas de experimentos, los campos por defecto, pudiendo modificarlos en cualquier momento.
- *View*: se ocupa de guardar las vistas creadas por el usuario, tanto para la gestión de experimentos como para la gestión de microarrays. Está vinculada a la tabla *Applicants* con una relación de n-1, puesto que cada vista solo puede pertenecer a un único usuario, y éste en cambio, puede tener n vistas. La tabla *View* también está vinculada con la tabla *Microarrays* en relación n-1, ya que del mismo modo, una vista solo puede pertenecer a una de las microarrays, y ésta por el contrario, puede tener n vistas.
- *Microarrays*: esta tabla se encargará de almacenar todas las microarrays que hayan sido dadas de alta por cualquier usuario. Está vinculada con la tabla de *Applicants* en relación n-1, ya que un usuario puede dar de alta a n microarrays, pero una microarray tan solo puede haber sido dada de alta por un usuario.
- *Genes*: se ocupa de guardar todos los genes de las microarrays que estén dadas de alta. Está vinculada con la tabla de *Applicants* en relación n-1, puesto que un usuario puede acceder a n genes, pero un gen solo puede haber sido dado de alta por un usuario; también está vinculada con la tabla *Microarrays* en relación n-1, ya que un gen solo pertenece a una microarray, pero ésta puede tener n genes.

Para identificar en la tabla de *View*, aquellas vistas que no correspondan a ninguna microarray en concreto, es decir, que pertenezcan a la gestión de experimentos, serán identificadas con el número 0, ya que como el índice de la tabla *Microarrays* es automático, el primer índice será siempre 1, con lo cual no habrá incongruencias en el sistema.

Cada microarray tendrá asociada una tabla *Experiments* específica, y será en dicha tabla donde se guardarán los datos y resultados de los experimentos realizados a partir de los datos de dicha microarray. Así, si en el futuro se quiere implementar una gestión, modificación o eliminación de microarrays diferente a la actual, la modularidad de la base de datos permitirá un acceso y una programación fácilmente realizable. Cabe añadir también, que dicha modularidad, hace que el trasvase de información para los futuros tratamientos de datos sea rápido y seguro.

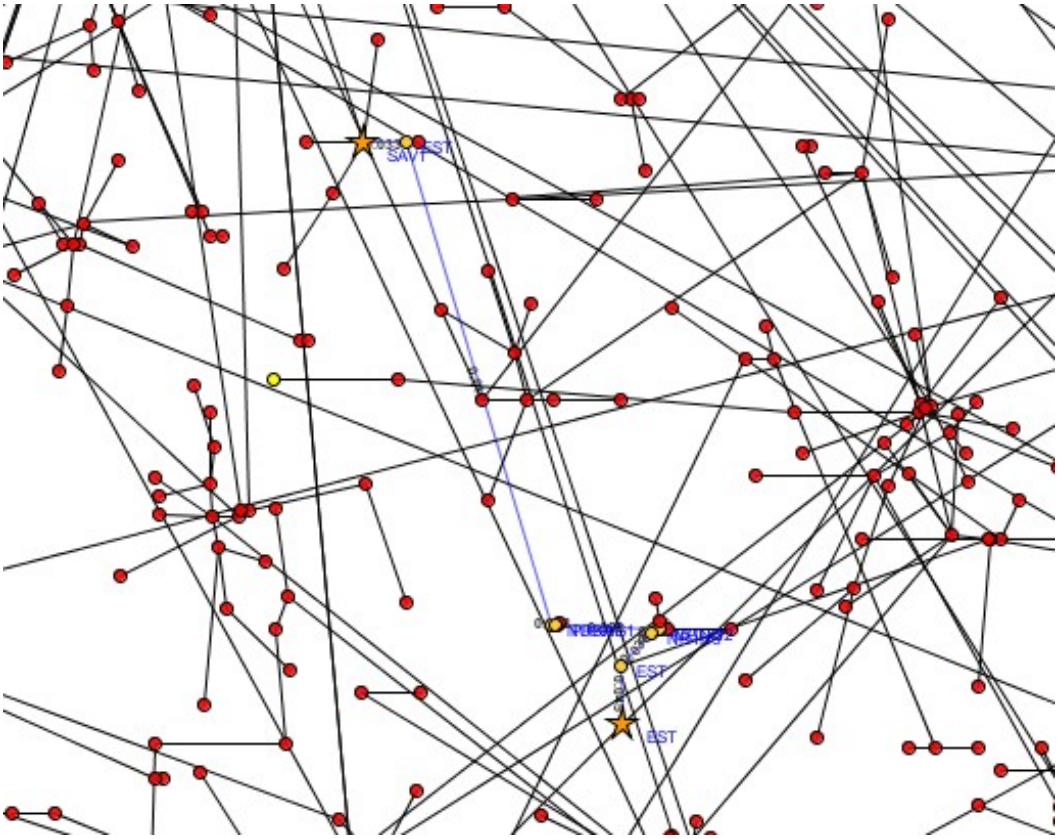
El diseño de la tabla de *Genes* ha sido pensado para que en el futuro, puedan añadirse unas funcionalidades concretas, como la modificación de los nombres de los genes según una plantilla específica, puesto que la mayoría de las veces, los genes no tienen un nombre específico sino un nombre alfa-numérico asignado temporalmente hasta la definición definitiva de uno.

5.3.- Pantallas de la Aplicación:

Veamos a continuación la funcionalidad de cada operación existente en la aplicación. Se analizará brevemente ya que dicha funcionalidad se encontraba creada y en funcionamiento y su comportamiento ya había sido objetivo de análisis y discusión en documentos anteriores a éste, que estaban centrados específicamente en detallarla.

Definiremos el comportamiento de cada función en este documento con el mero hecho de integrar todo el comportamiento de la aplicaicón en un documento.
Los ya existentes se describirán brevemente, centrándonos y analizando más profundamente la funcionalidad que nos interesa, la de cruces de datos genéticos empleando las bases de datos remotas.

El objetivo final de la aplicaicón es conseguir un conjunto de nodos (genes) seleccionados, después de realizar una serie de operaciones sobre el grafo, que nos permitan lanzar las aplicaciones externas para poder visualizar detalles relevantes sobre ellos. Un conjunto de genes seleccionados se visualizan en la aplicación como se ve en la figura siguiente:



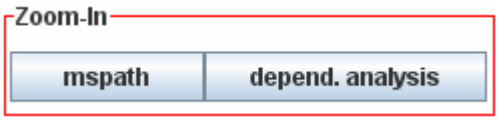
Se puede apreciar como hay genes seleccionados con una *shape* tipo estrella, otros que están en color naranja, que pueden ser el resultado de una operación como el "mspath", y otros de color amarillo que son genes seleccionados explícitamente por el usuario.

Este conjunto de genes pueden ser lanzados a una aplicación externa perfectamente y también pueden ser deseleccionados para poder hacer otra búsqueda nueva, haciendo click en cualquier parte del grafo con el Panel de Operaciones en *"Mode Picking"*, o es posible añadir manualmente alguno más pulsando la tecla *Shift*, y sin soltarla hacer click sobre uno de los genes no

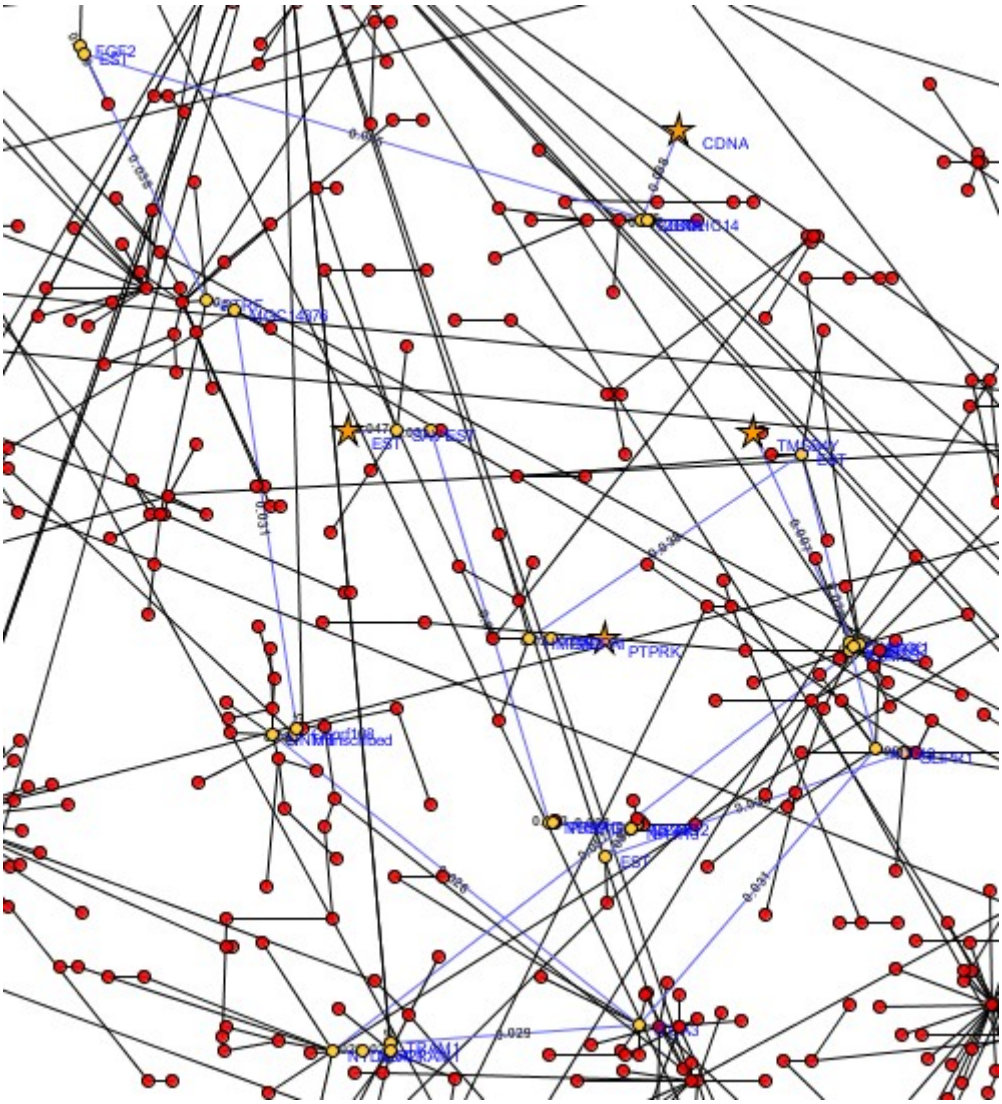
seleccionados. De la misma manera se puede deseleccionar un gen de los que están seleccionados.

Sabido el objetivo final, nos vamos a centrar en las operaciones, dividiendo éstas en bloques según su ubicación en el Panel de Operaciones.

Bloque zoom-in:

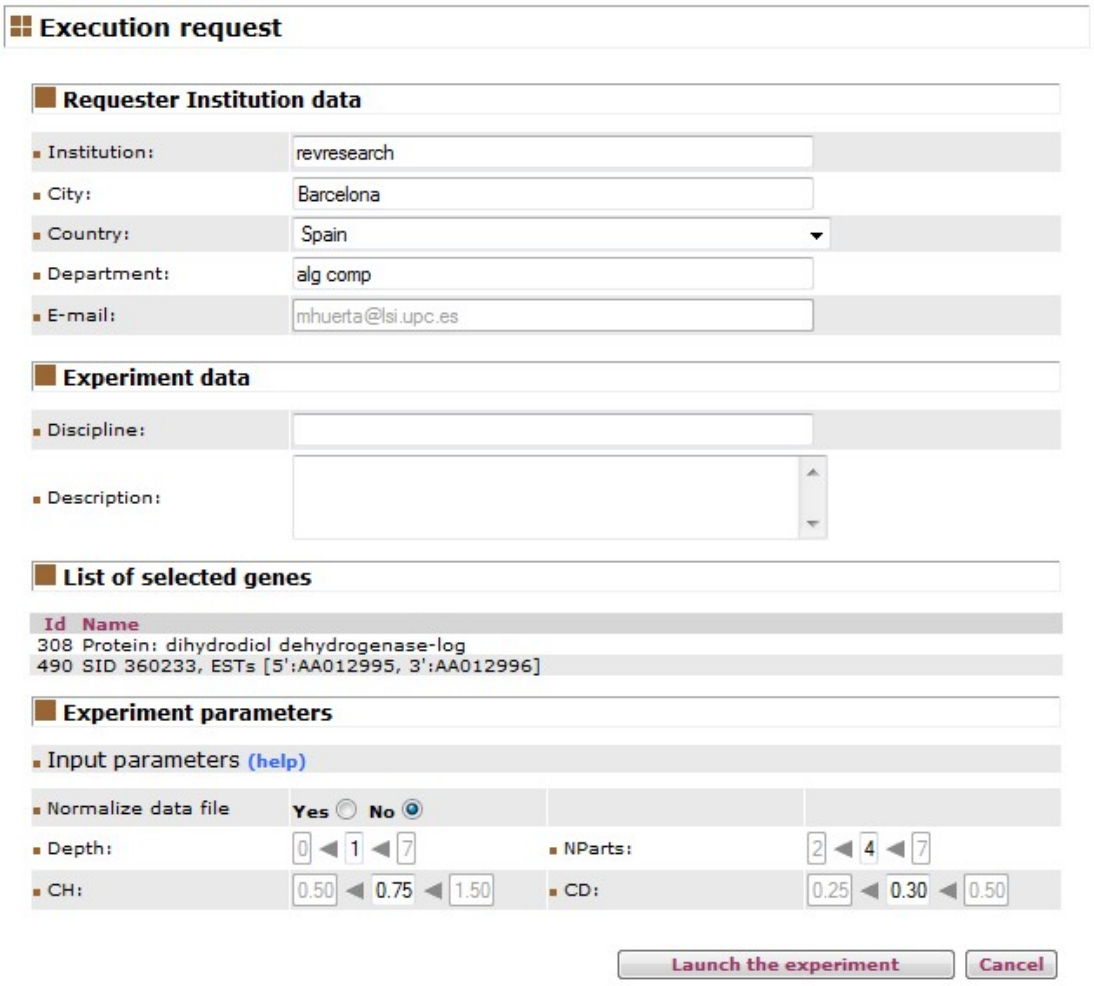


- **mspath:** Calcula el *Shortest Path* entre un grupo de genes seleccionados. Los genes seleccionados para la búsqueda del mspath son presentados en forma de Estrella de color naranja, y los nodos que forman parte del camino son presentados en color naranja suave.



Se puede apreciar como el camino se pinta de color azul, los nodos se presentan con la inicial del gen y como las correlaciones son mostradas encima de las aristas.

- **Dependence análisis:** Realiza el cálculo del patrón que define la dependencia en la expresión de los genes seleccionados. Es decir, el patrón de su relación para las muestras de la microarray cargada. Lanza la operación externa y se muestra la pantalla siguiente:

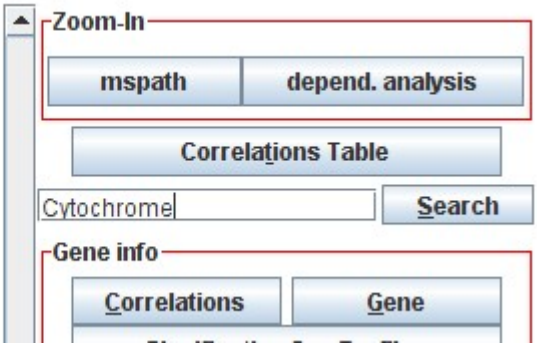


- **Correlations table:** Lanza la tabla de correlaciones entre todos los genes de la microarray.

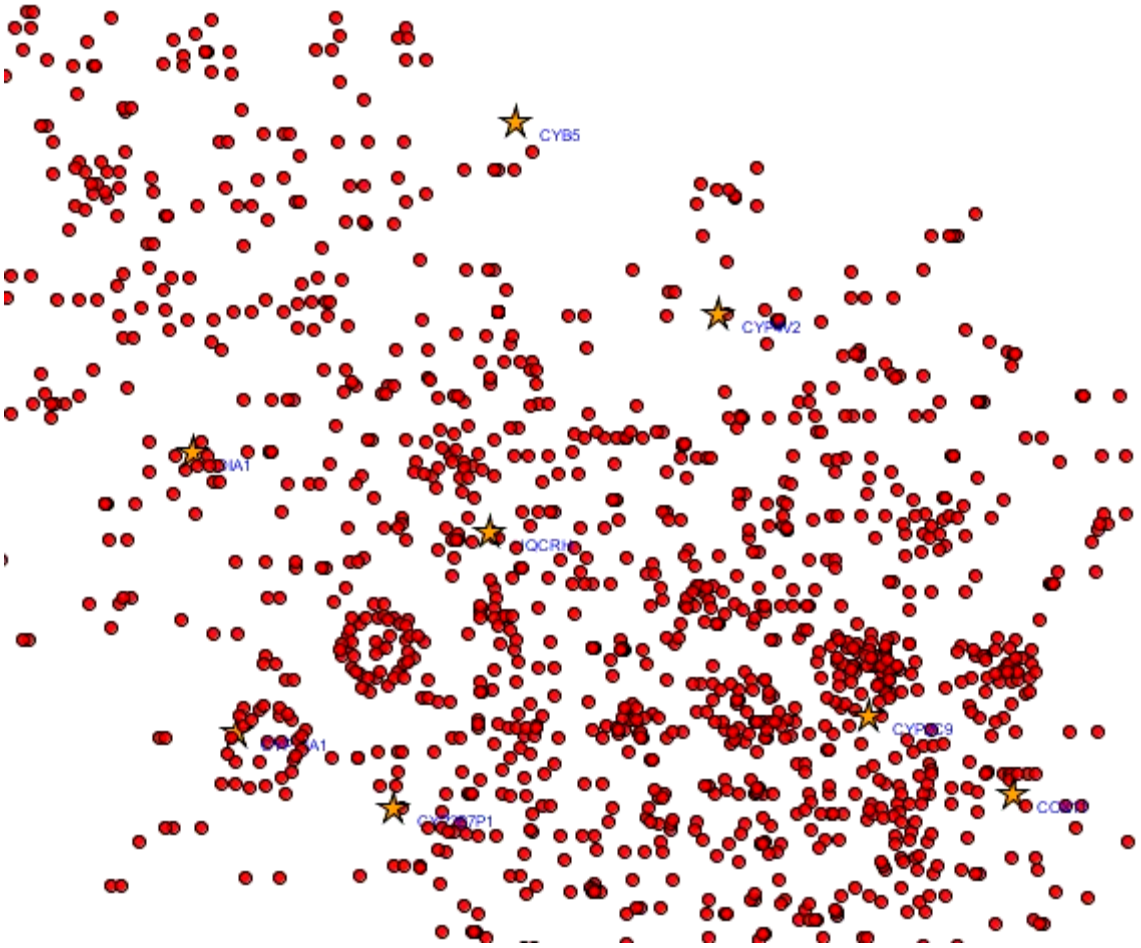
"at_matrix" :: gene-for-gene correlation table

		46 Protein	618 SID	1413 Protein	281 SID	4 Protein	895 SID	253 SID	252 ESTs,	1109 DESMOPL	430 ESTs	251 SID	248 SID	247 SID	250 ESTs	1100 H. sapie	965 *EST	256 Human	3 Human	243 SID	190 M-PHASE	1110 DESMOPL	1123 SID	994 Human	976 Protein	1349 SID	896 ESTs	309 LTBP1	249 ESTs	339 SID	254 SID
1	Protein:	0.01	0.01	0.02	0.04	0.04	0.05	0.05	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.08	0.08	0.08	0.08	0.08
2	SID	0.23	0.16	0.25	0.12	0.08	0.16	0.17	0.16	0.20	0.14	0.11	0.26	0.29	0.11	0.16	0.16	0.15	0.02	0.15	0.19	0.25	0.20	0.13	0.10	0.14	0.15	0.22	0.11	0.12	0.14
3	Human	0.22	0.17	0.22	0.12	0.08	0.34	0.14	0.13	0.25	0.14	0.10	0.14	0.19	0.10	0.16	0.15	0.19		0.15	0.19	0.17	0.23	0.11	0.08	0.13	0.13	0.18	0.09	0.09	0.15
4	Protein:	0.19	0.10	0.14	0.11		0.11	0.09	0.08	0.20	0.11	0.07	0.17	0.16	0.07	0.12	0.18	0.12	0.08	0.11	0.13	0.18	0.16	0.14	0.09	0.16	0.09	0.10	0.06	0.10	0.10
5	SID	0.18	0.15	0.23	0.12	0.03	0.15	0.14	0.12	0.32	0.20	0.08	0.20	0.20	0.10	0.32	0.20	0.15	0.09	0.13	0.17	0.25	0.23	0.11	0.10	0.14	0.49	0.20	0.08	0.09	0.13
6	ESTs	0.14	0.17	0.29	0.25	0.16	0.17	0.26	0.23	0.26	0.26	0.27	0.24	0.25	0.27	0.21	0.22	0.25	0.21	0.20	0.27	0.20	0.22	0.28	0.22	0.33	0.20	0.33	0.27	0.26	0.19
7	ESTs	0.24	0.18	0.29	0.25	0.17	0.22	0.26	0.28	0.31	0.28	0.26	0.25	0.36	0.25	0.34	0.22	0.25	0.16	0.29	0.27	0.36	0.33	0.23	0.20	0.20	0.20	0.35	0.23	0.23	0.26
8	Human	0.19	0.17	0.19	0.22	0.10	0.20	0.24	0.21	0.24	0.23	0.22	0.23	0.24	0.25	0.25	0.21	0.24	0.13	0.21	0.27	0.26	0.21	0.23	0.15	0.16	0.19	0.34	0.25	0.15	0.18
9	GALC	0.18	0.22	0.32	0.19	0.14	0.19	0.16	0.13	0.35	0.19	0.21	0.26	0.21	0.21	0.18	0.14	0.17	0.13	0.16	0.25	0.37	0.24	0.14	0.22	0.23	0.12	0.31	0.20	0.10	0.20
10	SID	0.27	0.15	0.26	0.16	0.12	0.13	0.22	0.21	0.31	0.16	0.16	0.19	0.18	0.15	0.19	0.10	0.18	0.09	0.17	0.29	0.29	0.26	0.19	0.20	0.17	0.22	0.31	0.12	0.15	0.22
11	INTERFERON-INDUCIBLE	0.23	0.23	0.31	0.16	0.14	0.15	0.19	0.23	0.26	0.16	0.16	0.18	0.19	0.19	0.22	0.18	0.17	0.14	0.16	0.34	0.35	0.27	0.26	0.22	0.22	0.13	0.26	0.15	0.18	0.23
12	ESTs	0.20	0.13		0.18	0.14		0.20	0.19		0.20	0.19	0.32	0.33	0.21			0.22	0.14	0.16	0.21							0.24	0.19	0.16	0.23
13	ESTs	0.20	0.18	0.27	0.22	0.19	0.21	0.26	0.21	0.32	0.22	0.21	0.28	0.26	0.21	0.17	0.16	0.22	0.12	0.18	0.19	0.28	0.24	0.20	0.21	0.19	0.20	0.21	0.19	0.19	0.25
14	Protein:	0.17	0.12	0.30	0.23	0.12	0.20	0.26	0.29	0.27	0.22	0.23	0.21	0.22	0.25	0.17	0.17	0.21	0.16	0.20	0.27	0.25	0.27	0.22	0.26	0.23	0.22	0.34	0.25	0.20	0.21
15	SID	0.30			0.17	0.08		0.20	0.17		0.18	0.12	0.19	0.21	0.12			0.22	0.17	0.14	0.28							0.20	0.11	0.26	0.17
16	SID	0.22	0.15	0.18	0.23	0.12	0.21	0.19	0.17	0.30	0.23	0.19	0.29	0.34	0.16	0.20	0.20	0.25	0.23	0.23	0.22	0.35	0.34	0.27	0.30	0.27	0.25	0.32	0.18	0.21	0.12
17	SID	0.24	0.13	0.14	0.16	0.08	0.14	0.27	0.24	0.22	0.25	0.20	0.24	0.24	0.18	0.20	0.26	0.19	0.12	0.16	0.23	0.23	0.25	0.26	0.27	0.17	0.12	0.32	0.18	0.18	0.11
18	SID	0.28	0.20	0.22	0.19	0.07	0.17	0.14	0.16	0.31	0.21	0.08	0.16	0.17	0.11	0.27	0.22	0.15	0.12	0.12	0.30	0.25	0.21	0.22	0.21	0.21	0.13	0.22	0.08	0.17	0.14
19	FBN2	0.23	0.11		0.18	0.09	0.13	0.22	0.23	0.36	0.17	0.13	0.23	0.23	0.14	0.24	0.19	0.22	0.12	0.16	0.28	0.39	0.28	0.21	0.19		0.13	0.23	0.15	0.20	0.17
20	Human	0.30	0.11	0.22	0.15	0.09	0.23	0.21	0.18	0.38	0.25	0.14	0.25	0.23	0.16	0.19	0.30	0.22	0.14	0.25	0.22		0.27	0.20	0.12	0.09	0.12	0.23	0.13	0.14	0.24
21	Human	0.22	0.12	0.16	0.17	0.11	0.22	0.18	0.15	0.30	0.18	0.15	0.27	0.24	0.17	0.18	0.20	0.26	0.22	0.26	0.14	0.29	0.27	0.19	0.17	0.21	0.18	0.22	0.16	0.16	0.21
22	ESTs	0.22	0.11	0.16	0.13	0.11	0.19	0.18	0.15	0.30	0.21	0.13	0.24	0.25	0.13	0.17	0.19	0.19	0.16	0.23	0.18	0.33	0.26	0.16	0.22	0.20	0.12	0.23	0.13	0.17	0.18
23	SID	0.19	0.10	0.15	0.13	0.10	0.16	0.15	0.12	0.28	0.17	0.11	0.22	0.17	0.12	0.15	0.16	0.18	0.13	0.18	0.18	0.30	0.21	0.17	0.17	0.21	0.10	0.25	0.11	0.14	0.16

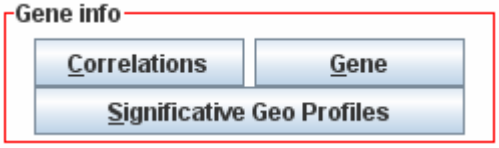
- **Search:** Realiza una búsqueda a partir del literal introducido en la *Input Box.*



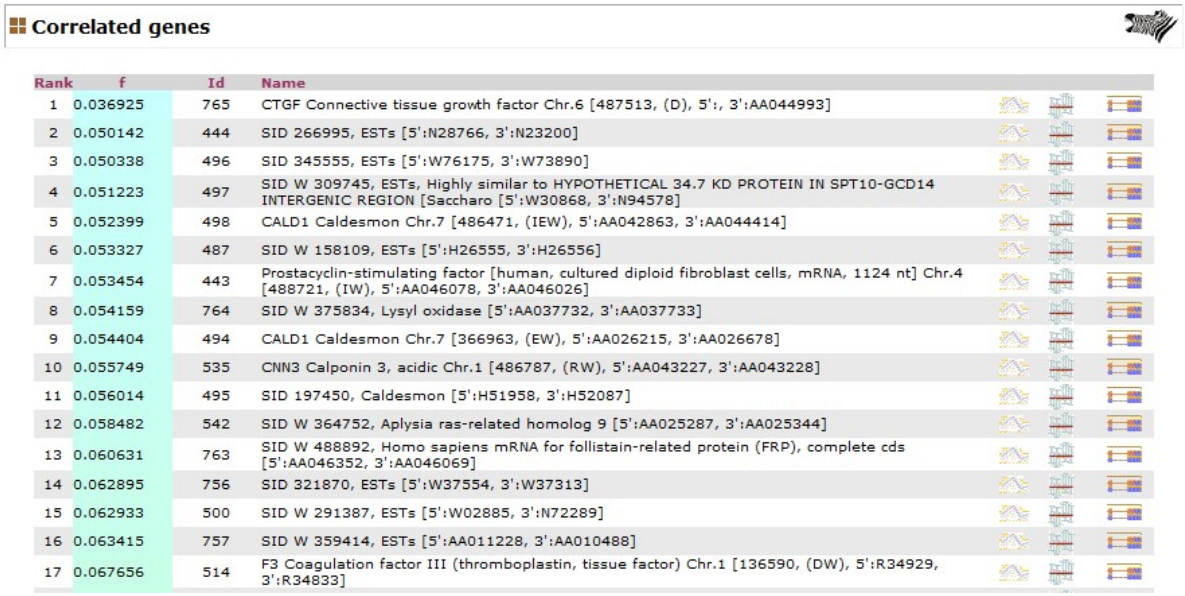
Si el Literal, exacto, forma parte de la *Label* del gen, este gen pasará a ser seleccionado. A continuación se muestran los genes encontrados, se resaltan con el *Shape* estrella naranja, en la búsqueda realizada para el literal ***Cytochrome***.



Bloque Gen info:

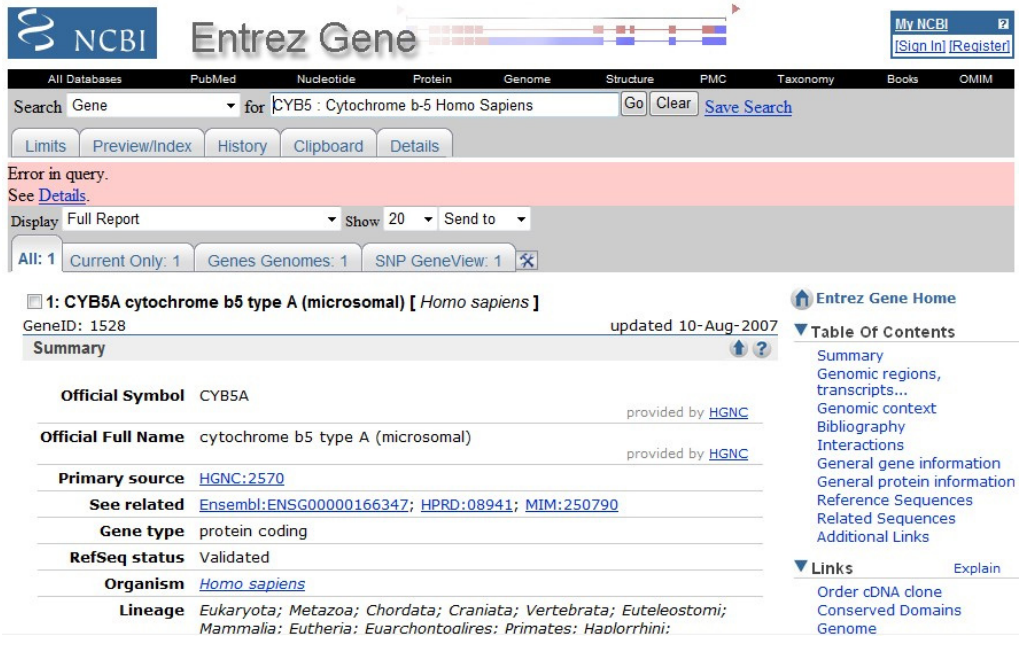


- **Correlations:** Lanza una consulta sobre el gen seleccionado y devuelve los genes más correlacionados con éste. Si hay más de un gen seleccionado lanza una pantalla por gen. En la página siguiente se puede observar la pantalla para uno de los genes encontrados en la operación descrita anteriormente.

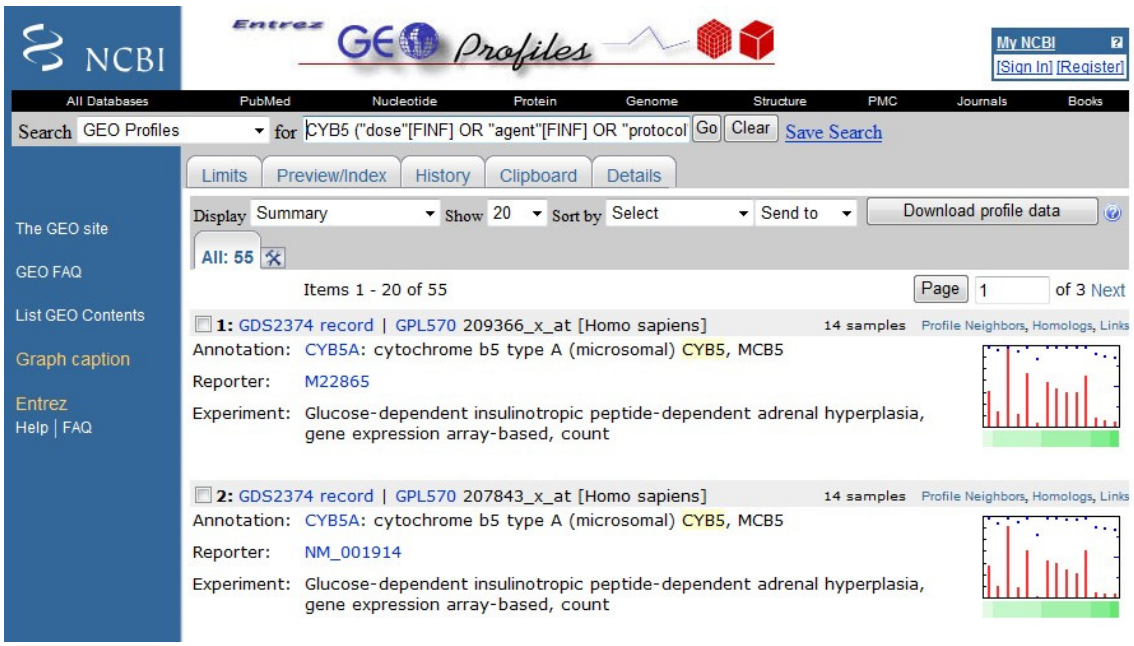


Rank	f	Id	Name
1	0.036925	765	CTGF Connective tissue growth factor Chr.6 [487513, (D), 5':, 3':AA044993]
2	0.050142	444	SID 266995, ESTs [5':N28766, 3':N23200]
3	0.050338	496	SID 345555, ESTs [5':W76175, 3':W73890]
4	0.051223	497	SID W 309745, ESTs, Highly similar to HYPOTHETICAL 34.7 KD PROTEIN IN SPT10-GCD14 INTERGENIC REGION [Saccharo [5':W30868, 3':N94578]
5	0.052399	498	CALD1 Caldesmon Chr.7 [486471, (IEW), 5':AA042863, 3':AA044414]
6	0.053327	487	SID W 158109, ESTs [5':H26555, 3':H26556]
7	0.053454	443	Prostacyclin-stimulating factor [human, cultured diploid fibroblast cells, mRNA, 1124 nt] Chr.4 [488721, (IW), 5':AA046078, 3':AA046026]
8	0.054159	764	SID W 375834, Lysyl oxidase [5':AA037732, 3':AA037733]
9	0.054404	494	CALD1 Caldesmon Chr.7 [366963, (EW), 5':AA026215, 3':AA026678]
10	0.055749	535	CNN3 Calponin 3, acidic Chr.1 [486787, (RW), 5':AA043227, 3':AA043228]
11	0.056014	495	SID 197450, Caldesmon [5':H51958, 3':H52087]
12	0.058482	542	SID W 364752, Aplysia ras-related homolog 9 [5':AA025287, 3':AA025344]
13	0.060631	763	SID W 488892, Homo sapiens mRNA for follistain-related protein (FRP), complete cds [5':AA046352, 3':AA046069]
14	0.062895	756	SID 321870, ESTs [5':W37554, 3':W37313]
15	0.062933	500	SID W 291387, ESTs [5':W02885, 3':N72289]
16	0.063415	757	SID W 359414, ESTs [5':AA011228, 3':AA010488]
17	0.067656	514	F3 Coagulation factor III (thromboplastin, tissue factor) Chr.1 [136590, (DW), 5':R34929, 3':R34833]

- **Gene:** Lanza la información de la base de datos del NCBI. Si hay más de un gen seleccionado lanza una pantalla para cada gen. La imagen siguiente muestra la información recogida en el NCBI para un gen encontrado en la operación "Search":



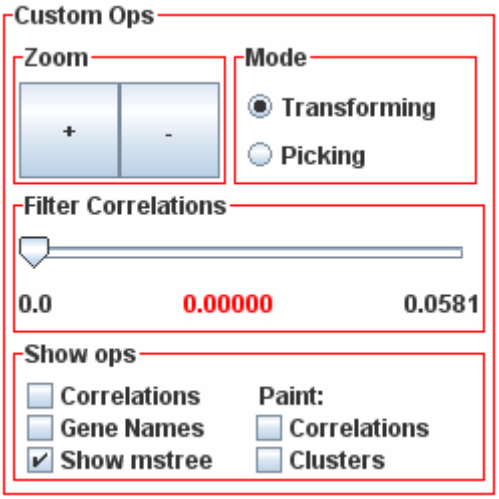
- **Significative Geo Profiles:** Lanza una consulta a la base de datos de *microarray datasets* del NCBI, devolviendo las Microarrays con las que el gen seleccionado dé contrastes relevantes para las muestras de esas Microarrays. La pantalla siguiente muestra para el gen anterior la pantalla comentada:



- **Update Samples-Class Palete:** Lanza un applet para cambiar la paleta de colores asociada a la microarray. Es decir, agrupa los diferentes *samples* de la microarray en colores de forma que se puedan distinguir dichos grupos cuando se visualizan las dependencias entre genes.

- **Class –Distribution Gene Search:** Lanza una búsqueda de los genes que se expresan de determinada forma para los grupos definidos para la *Samples-Class Palette*.

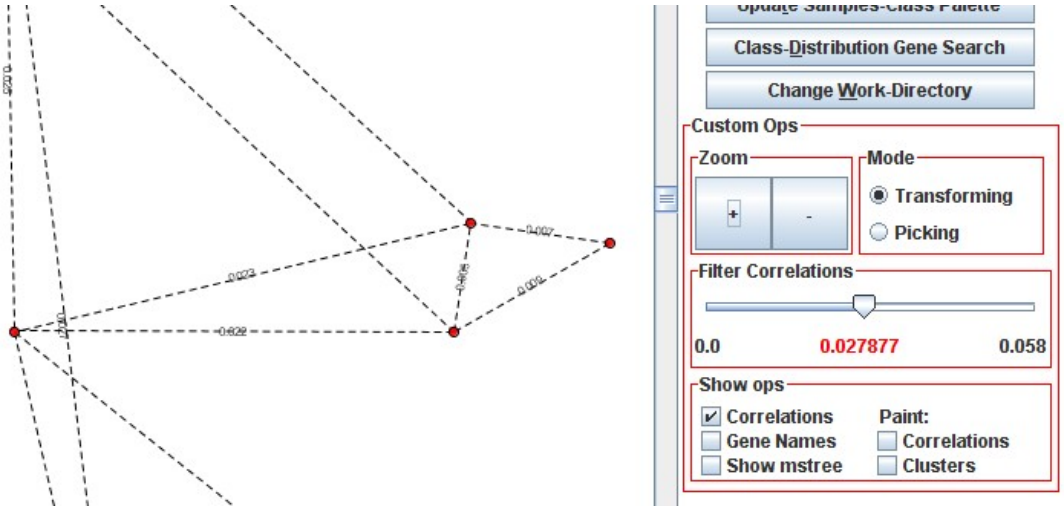
Bloque Custom Operations:



- **Zoom:** Realiza un zoom sobre la ventana de visualización del grafo. Si el mouse dispone de rueda, la manipulación de esta también aplica un zoom sobre la ventana de visualización.

- **Mode:** Permite especificar el modo en el que el mouse actúa sobre el grafo. Tiene dos opciones:
  1. **Transforming:** Mueve y arrastra el grafo en su totalidad a través de la ventana de visualización.
  2. **Picking:** Sirve para seleccionar elementos del grafo, tanto nodos como aristas. Si se hace clic sobre la tecla ***Shift*** **,** y sin soltar ésta, la operación es acumulativa.

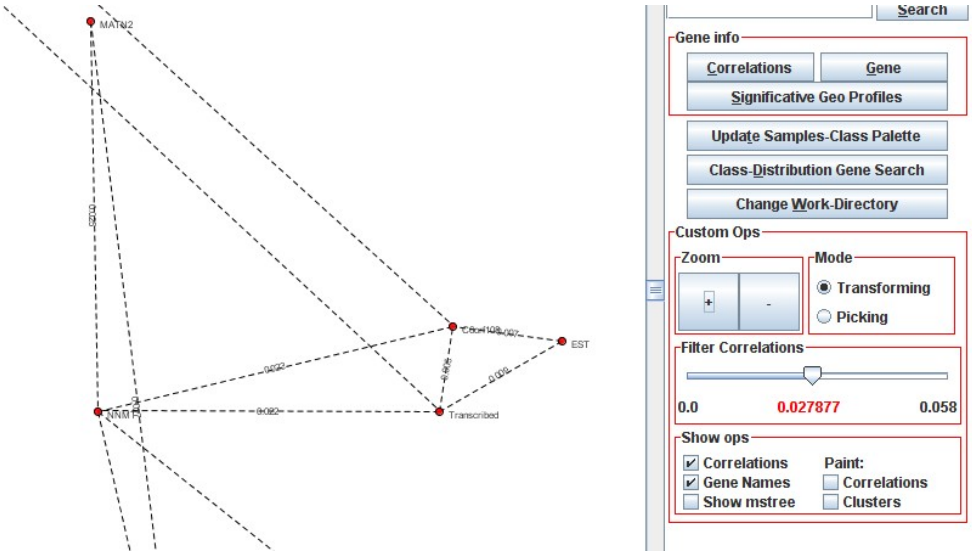
- **Filter Correlations:** Control que es un *Java.Swing.JSlider* que permite aplicar un filtro de correlaciones sobre el grafo permitiendo visualizarse sólo las aristas que tengan un valor menor o igual al escogido con el control. En la figura siguiente se puede observar la aplicación del filtro en el grafo realizando un zoom en una región de éste.



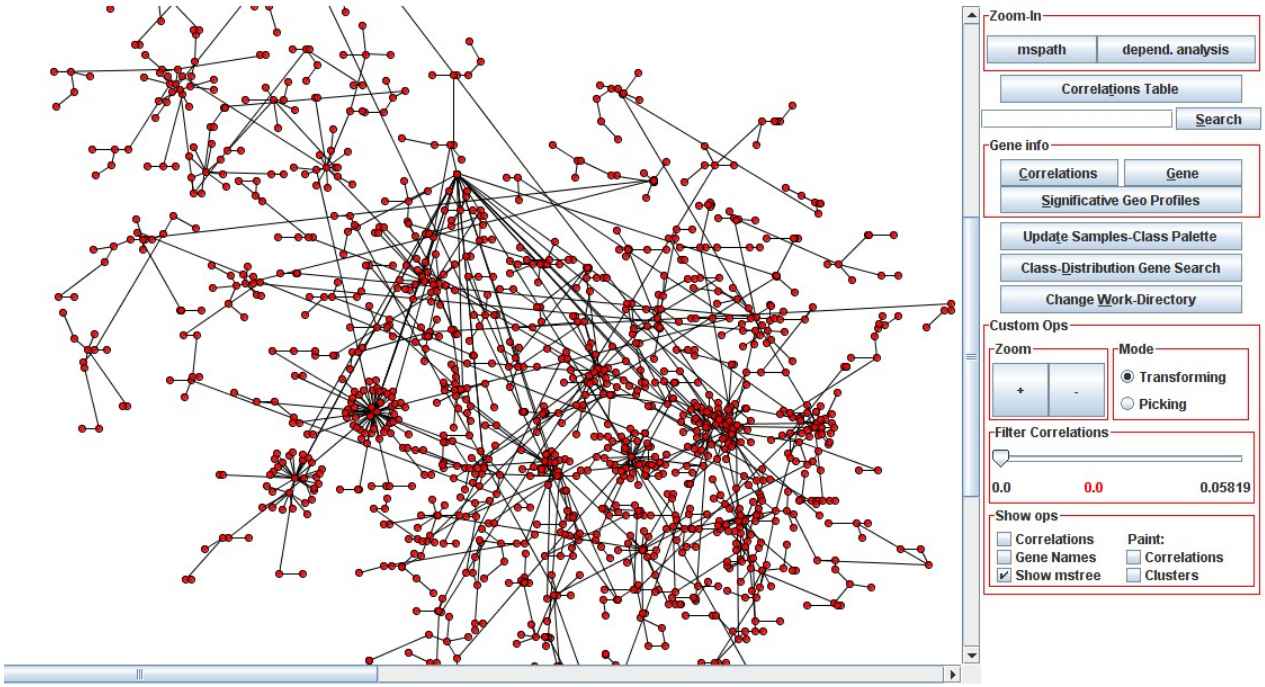
- **Show Ops:** Este bloque de operaciones sirve para visualizar en detalle elementos y entidades del grafo. Son:

    1. **Correlations:** Muestra en cada arista el valor de dicha correlación. En la figura anterior se puede observar un ejemplo de esta operación conjuntamente con la de filtro.

    2. **Gene Names:** Muestra las etiquetas de los genes. Aprovechando el ejemplo anterior y haciendo clic sobre *Gene Names* tenemos la siguiente pantalla de salida:

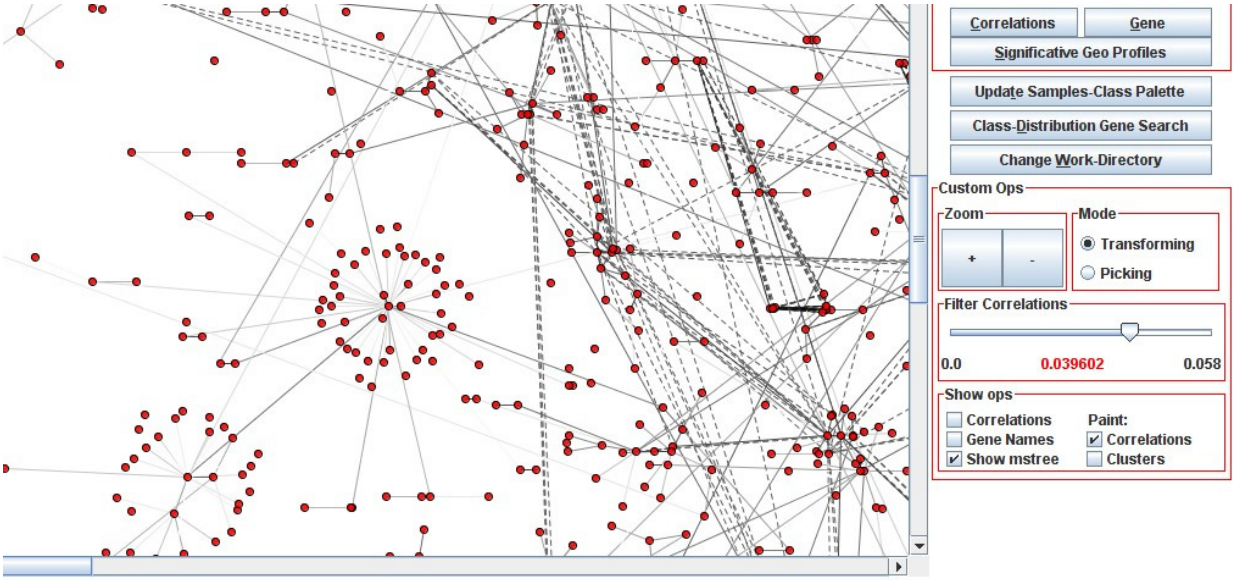


    3. **Show mstree:** Muestra el *minimum spanning tree* asociado al grafo y calculado en el preproceso.

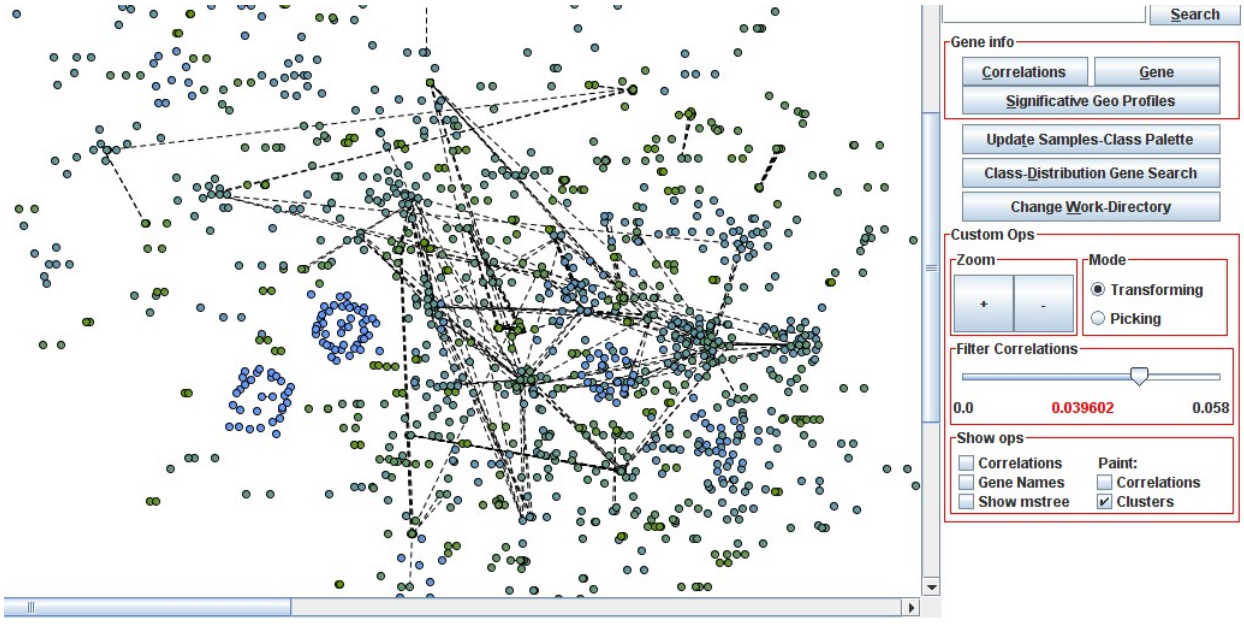


4. **Paint:** Bloque que sirve para pintar las aristas o los nodos de una manera predeterminada y que sirve para hallar información relevante en el grafo. Son dos operaciones de coloración del grafo:

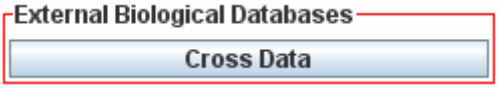
    - **Correlations:** Asigna un nivel de gris a cada arista equivalente al valor de correlación que ésta posee. La pantalla siguiente muestra cual es el resultado de pintar las correlaciones sobre el grafo filtrado y con la operación mstree realizada. Se aprecia en el zoom de la región del grafo como existen aristas con un color gris más tenue y otras con un gris más acentuado.



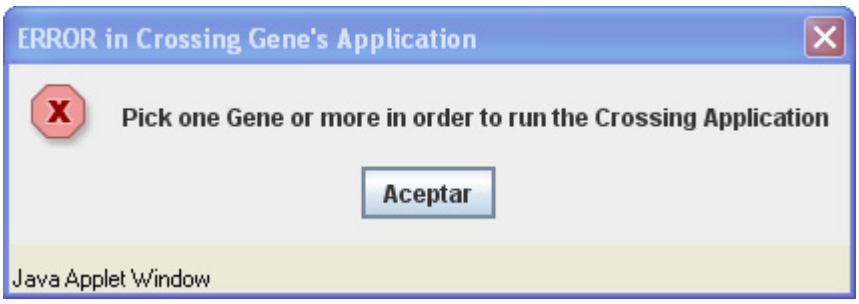
    - **Clusters:** Asigna un color a cada nodo. En la pantalla siguiente se pude apreciar como se detectan clusters de nodos con una dependencia entre si muy evidente.



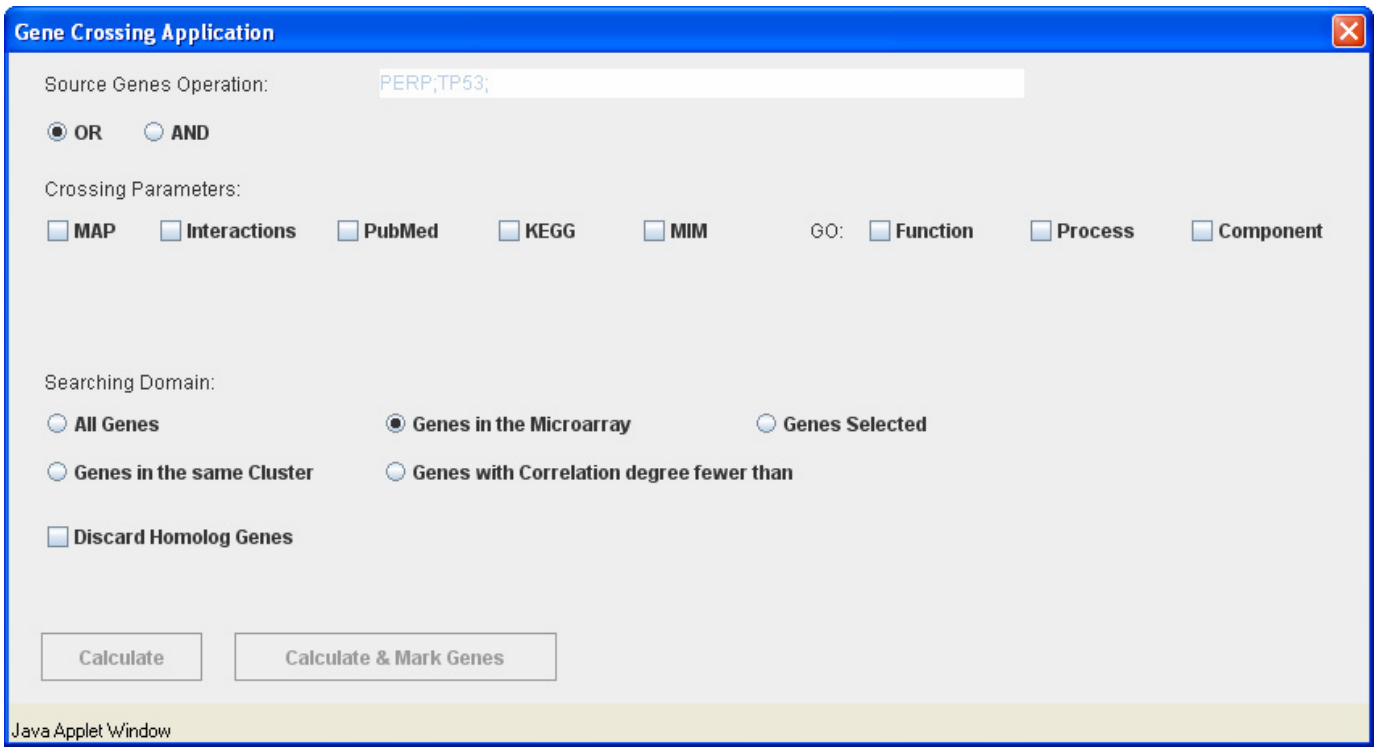
5.3.1.- Bloque External Biological Databases. Funcionalidad de cruces:



- **Cross Data:** Este botón inicia la aplicación de cruce de datos que hemos desarrollado en este proyecto fin de carrera. Su principal funcionalidad es la de obtener una serie de genes cómo resultado de consultas a bases de datos remotas. Para poder comenzar la aplicación, es necesario tener al menos un gen origen marcado en el grafo, de manera que si no hay ninguno marcado, se mostrará un mensaje de error:

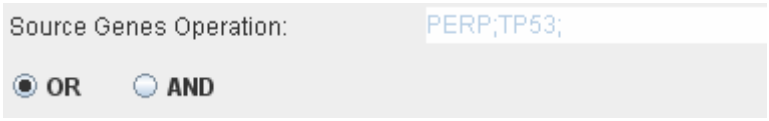


Si al menos hay un gen seleccionado, se lanza la ventana de cruce de datos. En ella puede observarse la totalidad de opciones que el usuario puede seleccionar a la hora de decidir qué resultados desea, y que bases de datos remotas consultar:



Estudiemos con detenimiento cada opción de esta pantalla.

En la parte superior aparecen los símbolos de los genes que se han seleccionado como entrada a la aplicación. Esta información no será modificable, su única funcionalidad es informar al usuario sobre qué genes se va a realizar el proceso de cruce de datos:



En este caso, como ejemplo estamos utilizando dos genes, el PERP y el TP53.

Además podrá elegir si realizar un AND u OR de los genes orígenes:

- OR: mostrar genes solución para cada gen origen.
- AND: mostrar genes solución comunes para todos los genes orígenes.

Durante el transcurso de la explicación vamos a tener en cuenta que se encuentra marcada la casilla *OR*. Si se marcara la *AND*, se obtendrían sólo resultados comunes a todos los genes origen *(Posteriormente veremos una captura)*.

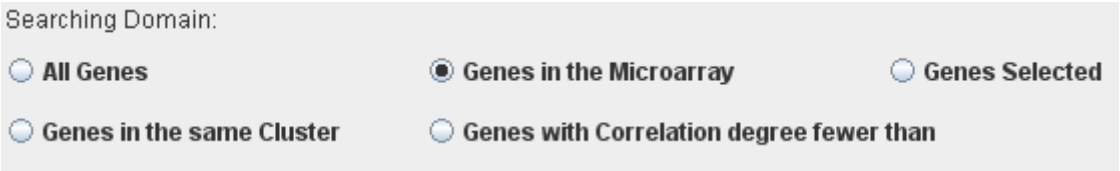
A continuación se ha de elegir las bases de datos sobre las que cruzar información (PubMed, KEGG, OMIM, Interactions, GO, Locations…).

Dependiendo de las bases de datos elegidas, se realizarán los cruces en base a unos parámetros u otros:

- **MAP**: se obtendrán genes vecinos a los genes orígenes.
- **Interactions**: se obtendrán genes que interaccionan con genes orígenes.
- **PubMed**: en este caso se obtendrán genes que aparecen en mismas publicaciones PubMed que los genes origen. Si seleccionamos esta base de datos, aparecen tres campos más:
  - Campo de texto para indicar frase a buscar en PubMed (*Ej: genes que aparezcan en PubMeds que traten sobre cancer*).
  - Casilla para elegir como resultado aquellos PubMeds de los genes orígenes que tengan la misma taxonomía que la microarray (*Human, Rat,…*).
  - Casilla para elegir como resultado aquellos PubMeds de los genes orígenes de cualquier taxonomía existente (*Ej: si el gen origen es TP53, se tomarán todas las publicaciones de todos los genes TP53 de todas las especies*).
- **KEGG:** se obtendrán genes que se encuentren en mismos mapas de Pathways que los genes orígen.
- **MIM:** genes que presenten mismas patologías que genes orígen.
- **Function:** genes presentes en mismas funciones celulares que los genes orígen.
  - Se incluye campo de texto para filtrar por función si se desea (*Ej: genes que tengan mismas funciones que los orígenes y ésta sea ATP Binding*). Si no se incluye nada, se toman todas las funciones comunes.

- **Process:** genes presentes en mismos procesos celulares que los genes orígen.
    - Se incluye campo de texto para filtrar por proceso si se desea (*Ej: genes que tengan mismos procesos que los orígenes y éste sea Protein Complex Assembly*). Si no se incluye nada, se toman todos los procesos comunes.
- **Component:** genes presentes en mismos componentes celulares que los genes orígen.
    - Se incluye campo de texto para filtrar por componente si se desea (*Ej: genes que se encuentren en mismos componentes que los orígenes y éste sea Membrane (membrana celular)*). Si no se incluye nada, se toman todos los componentes comunes.

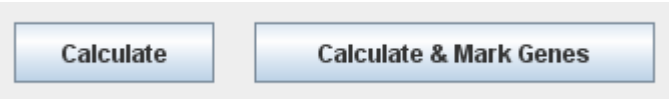
La siguiente opción a seleccionar es el dominio de búsqueda sobre el que se van a obtener los genes resultado:



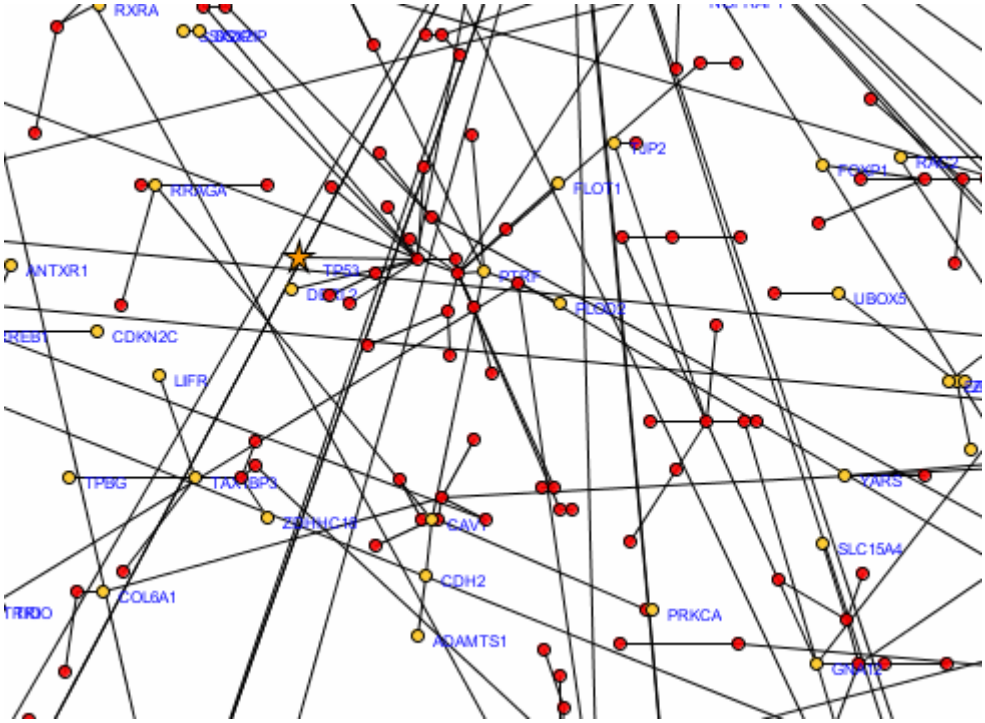
- Genes seleccionados – Se buscarán los genes resultado de entre aquellos que estén seleccionados en el grafo de la microarray y sean a su vez origen de la consulta.
- Genes del mismo cluster que los seleccionados – Para obtener estos genes se ha de leer del fichero *BioFACTORS2K5.net* los clusters en los que están los genes orígenes.
- Genes con determinado grado de correlación que los seleccionados – Existe una serie de ficheros dentro del directorio de la microarray en los que se almacenan los grados de correlación de cada gen con el resto de genes de la microarray.
- Genes en la microarray – Se tomarán como resultado aquellos genes que estén en la microarray.
- Todos los genes. Se obtendran los resultados de entre todos los genes existentes hasta el momento. Para este último caso, sólo se pintarán en el grafo como solución los genes que estén en la microarray actual. El resto de genes aparecerán en la página web de resultados.

El siguiente paso es elegir si se desea hacer un filtro de los genes resultados, desechando aquellos que sean homólogos entre si. Para ello, se consultará remotamente vía eUtils a la base de datos de Homologene, comprobando para cada gen resultado sus homólogos y, si se encuentran entre los genes solución, desecharlos para dejar sólo uno.

Cómo último paso es seleccionar de qué forma deseamos los resultados:

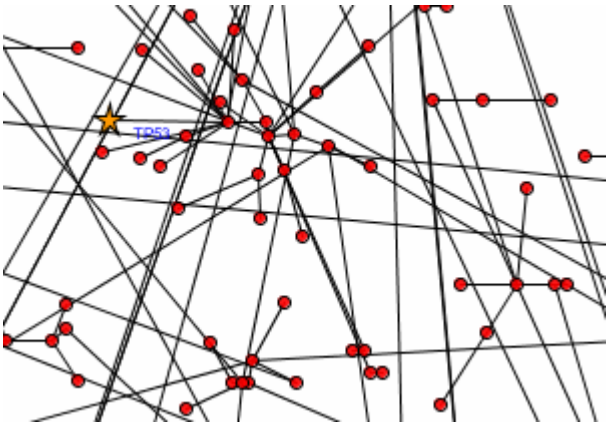


- **Calculate** – Se mostrarán los genes solución en un entorno web que ahora pasaremos a explicar.
- **Calculate & Mark Genes** – Además de mostrar los genes solución en web, se marcarán en el grafo de la microarray. Los genes orígenes de la operación de cruces aparecerán marcados con una estrella naranja y los solución aparecerán marcados con un circulo amarillo:



Todos los genes que aparecen seleccionados en el grafo serán la entrada para la siguiente operación que se realice en la aplicación, bien sea una nueva de cruce de datos, o una de obtener el MSPath, o información de los genes…

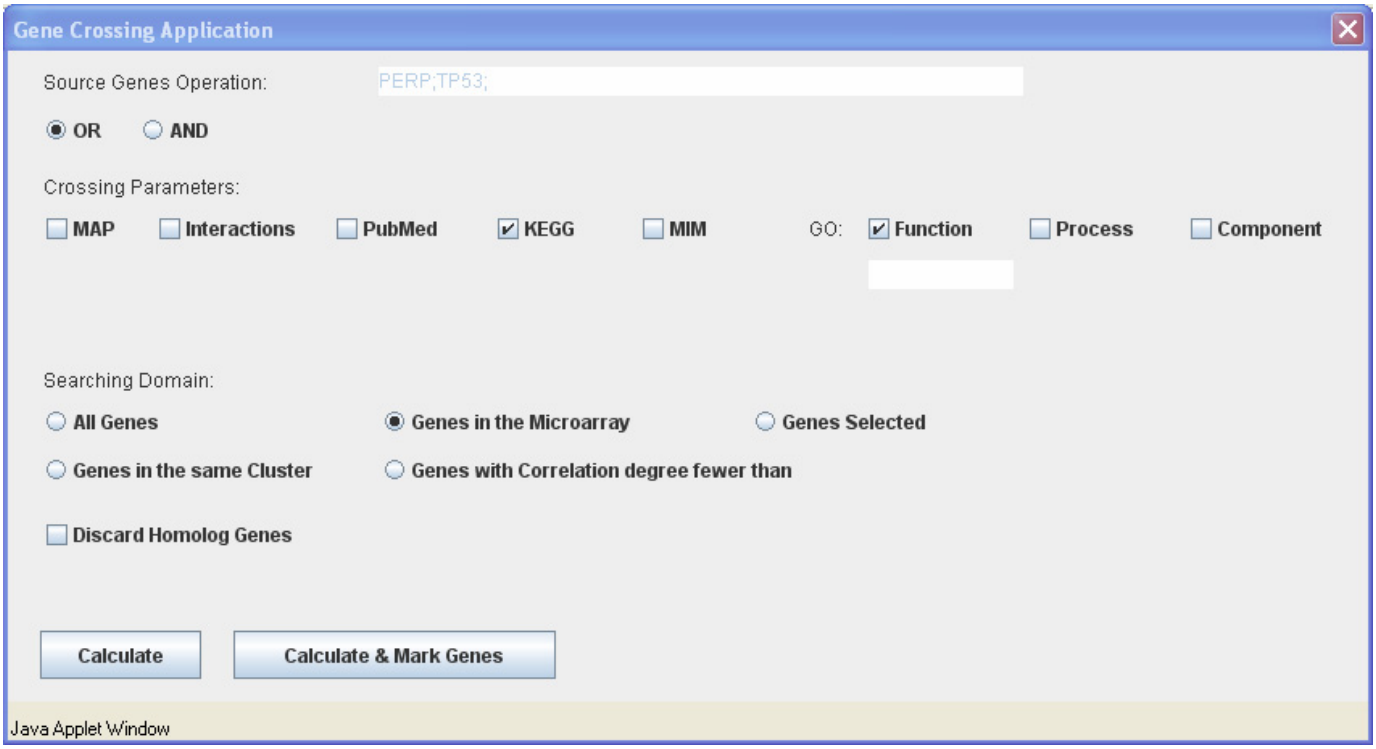
El botón calculate no marca ningún gen resultado, tan sólo seguiran picados los genes iniciales tal cual estaban:



Lista de Resultados finales:

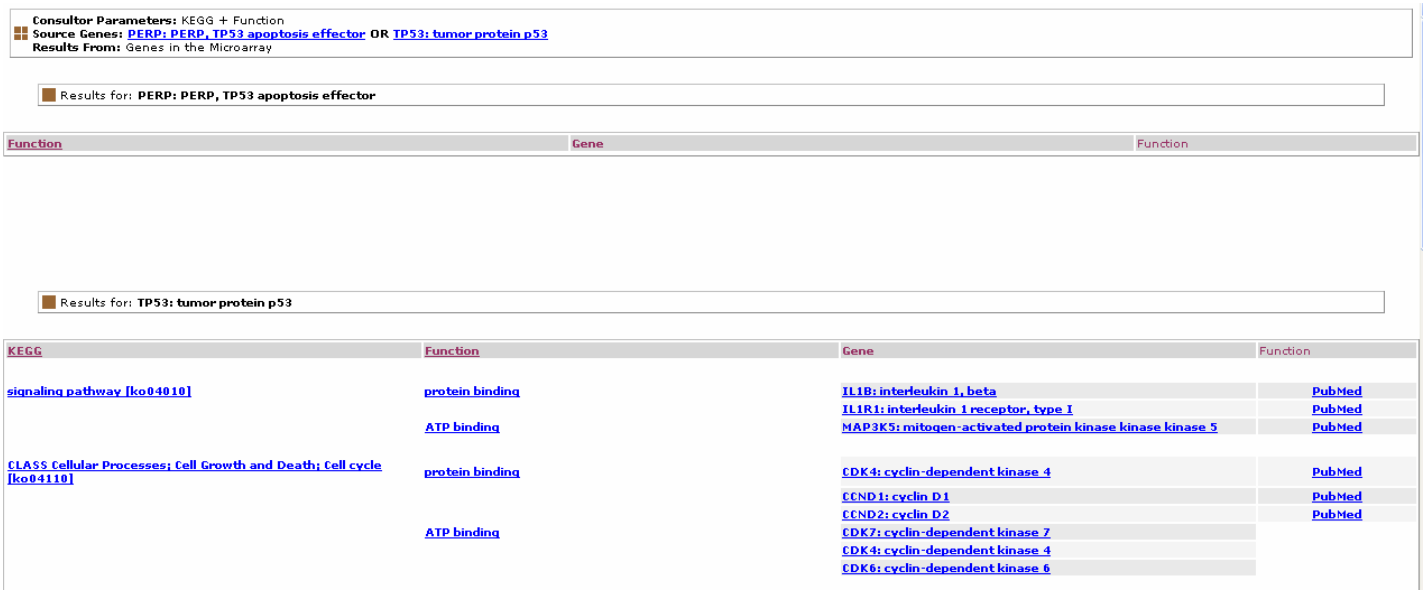
En ambos casos, picando el botón *Calculate* o *Calculate & Mark Genes* se generará una página web con los resultados de la operación.

Analicemos los resultados obtenidos para la siguiente operación de ejemplo:



En este caso deseamos como solución los genes **de la microarray** que aparezcan en mismo **KEGG** y mismas **funciones** que los genes orígenes (PERP y TP53), empleando la opción **OR** (resultados independientes para cada gen de origen).

La página generada como resultado es la siguiente:



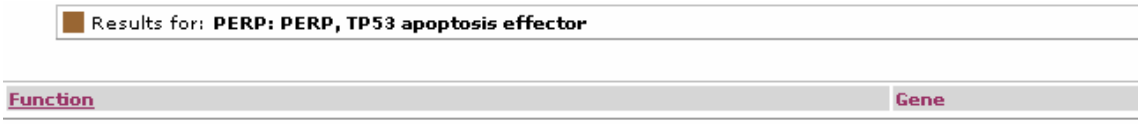
Analicémosla con más detalle:

- En primer lugar aparece una cabecera de información, en la cual se indican los genes orígenes de la aplicación de cruces, las bases de datos seleccionadas con sus parámetros, el dominio de búsqueda y si se emplea filtro de homólogos:

  En este caso observamos que los parámetros de búsqueda son KEGG y Function, los genes orígenes PERP y TP53, la operación es un OR y los resultados se toman de los genes de la microarray.

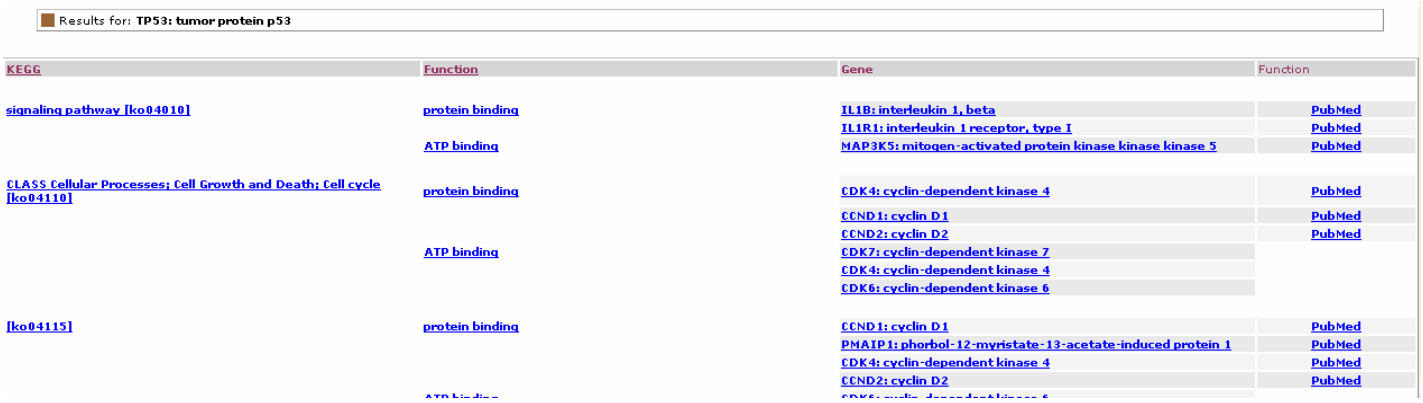
  El enlace a cada gen hace más facil la búsqueda de resultados para dicho gen dentro de la propia página. Este enlace es útil cuando los resultados son largos y queremos acceder directamente a los resultados de un gen determinado. No es más que un *ancla html* dentro de la propia página.

- En segundo lugar aparecerá una cabecera del primer gen origen. Bajo esta cabecera se muestran los resultados obtenidos para dicho gen, según los parámetros de búsqueda:



  En este caso observamos que no existe ningún resultado para el gen origen PERP.

  La siguiente cabecera corresponde al segundo gen origen, TP53 en este caso, que si tiene resultados a la consulta:



  Aparecen 4 columnas. Las dos primeras corresponden a los resultados para las bases de datos elegidas. La tercera es el nombre del gen resultado y la cuarta es el enlace a la publicación que asigna ese gen resultado a la función en la que aparece junto a TP53.

KEGG	Function
signaling pathway [ko04010]	protein binding
	ATP binding

Vemos que para el KEGG *signaling pathway*, y funciones *Protein Binding y ATP binding* hay tres genes:

Gene	Function
IL1B: interleukin 1, beta	PubMed
IL1R1: interleukin 1 receptor, type I	PubMed
MAP3K5: mitogen-activated protein kinase kinase kinase 5	PubMed

El primer y segundo gen son los resultados que está en KEGG *Signaling pathway* y función *Protein Binding*. El tercero está en el mismo KEGG pero en la función *ATP Binding*.

En este caso estamos ordenando por KEGG, es decir, todos los resultados aparecen ordenados primero por KEGG y luego, dentro de cada KEGG, por función.
Si quisiéramos ordenarlos por Function, tan sólo tendríamos que pinchar en el enlace *Function* existente en la columna.

Function	KEGG	Gene
protein binding	signaling pathway [ko04010]	IL1B: interleukin 1, beta
		IL1R1: interleukin 1 receptor, type I
	CLASS Cellular Processes; Cell Growth and Death; Cell cycle [ko04110]	CDK4: cyclin-dependent kinase 4
		CCND1: cyclin D1
		CCND2: cyclin D2
	[ko04115]	CCND1: cyclin D1

Ahora vemos que aparecen los genes ordenados primero por Function y luego, dentro de cada función por los KEGG existentes.

Si hubiera más de dos bases de datos, el orden sería ascendente. Es decir, al pinchar en el título de cada columna, ascendería una posición en la jerarquía, excepto al pinchar en la columna de primer orden, que bajaría una posición.

Ejemplo: Si tuviéramos el siguiente resultado:

KEGG Function Process

Y pinchamos en Process, el orden sería:

KEGG Process Function

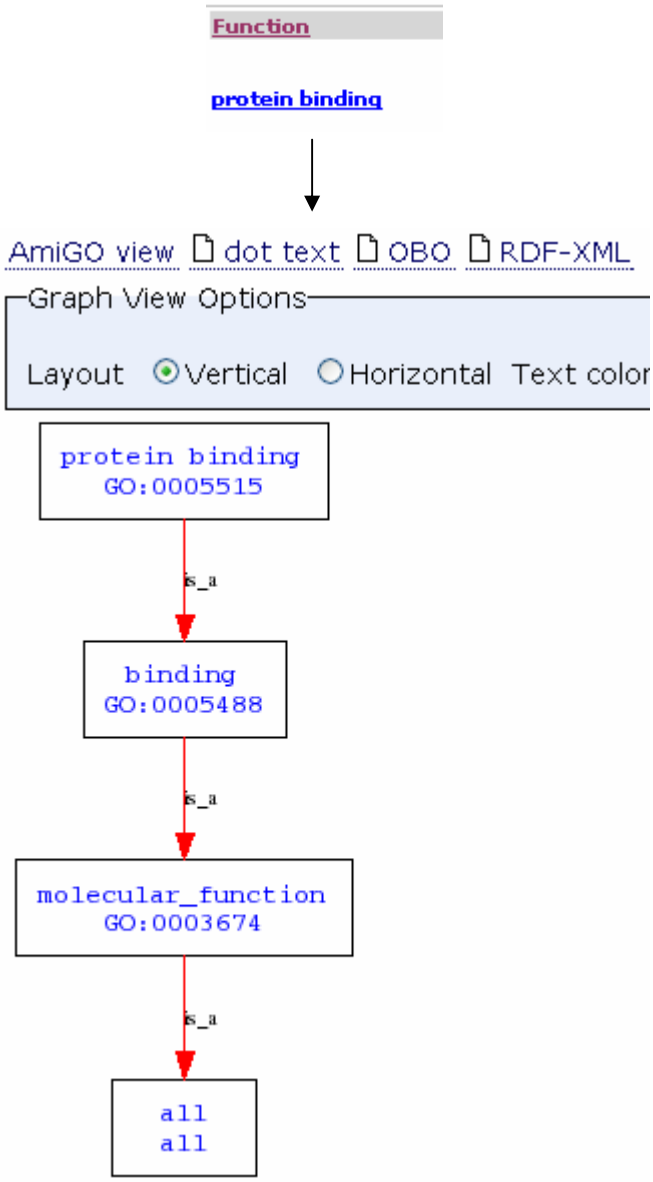
Si a continuación pinchamos en KEGG, el nuevo orden sería:

Process KEGG Function

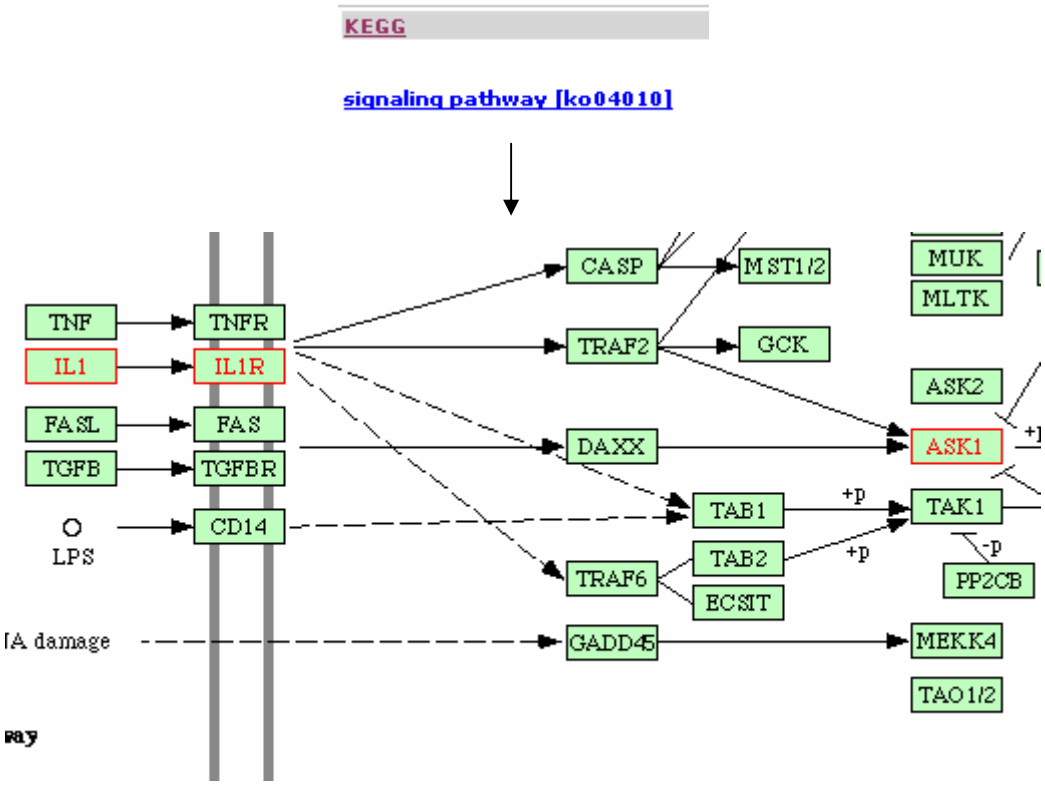
Cada resultado mostrado es un enlace a una página del NCBI o del KEGG que complementa la información resultante, dotándola de toda la significación posible para el usuario. Así, esta página resultado pretende ser un centro de información de aquellos genes que se han obtenido como resultado del cruce de datos.

Los enlaces para cada resultado de base de datos son los siguientes:

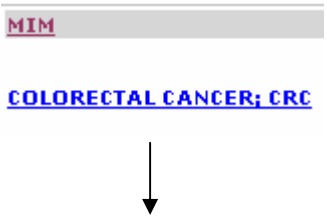
- **Function, Process y Component**: Enlace a un gráfico de la página del GO:



- **KEGG**: Enlace a un gráfico de mapa del KEGG, pintando de un color diferente los genes resultados de la consulta:



- **MIM**: Enlace a la información de la patología en cuestión:



#114500 GeneTests, Links

COLORECTAL CANCER; CRC

Alternative titles; symbols

COLON CANCER

Gene map locus 20q13.2-q13.3, 18q21.1, 17q24, 17p11.2, 17p13.1, 15q15, 14q32.3, 14q24.3, 11p11.2, 9q32-q33, 1p13.2, 8p22-p21.3, 5q21-q22, 4q32, 3q26.3, 1p35, 2p25, 22q13

TEXT

A number sign (#) is used with this entry because mutations in several different genes have been identified in colorectal cancer. Mutations in a single gene result in a marked predisposition to colorectal cancer in 2 distinct syndromes: familial adenomatous polyposis (FAP; 175100) and hereditary nonpolyposis colorectal cancer (HNPCC; see 120435). FAP is caused by mutations in the APC gene (611731), whereas HNPCC is caused by mutations in several genes, including MSH2 (609309), MLH1 (120436), PMS1 (600258), PMS2 (600259), MSH6 (600678), TFGBR2 (190182), and MLH3 (604395). Other colorectal cancer syndromes include autosomal recessive adenomatous polyposis (608456), which is caused by mutations in the MUTYH gene (604933), and oligodontia-colorectal cancer syndrome (608615), which is caused by mutations in the AXIN2 gene (604025). The CHEK2 gene (604373) has been implicated in susceptibility to colorectal cancer in Finnish patients. In addition, somatic mutations in many different genes, including KRAS (190070), PIK3CA (171834), BRAF (164757), CTNNB1 (116806), AXIN2 (604025), AKT1 (164730), and MCC (159350), have been identified in colorectal cancer. Genomewide association studies have shown that common alleles of SMAD7 (602932) influence colorectal cancer risk.

Susceptibility loci for colorectal cancer have been mapped to chromosomes 9q22.2-q31.2 (CRCS1; 608812) and 8q24 (CRCS2; 611469).

CLINICAL FEATURES

Colon cancer is a well-known feature of familial polyposis coli. Cancer of the colon occurred in 7 members of 4 successive generations of the family reported by Kluge (1964), leading him to suggest a simple genetic basis for colonic cancer independent of polyposis. The combination of colonic and endometrial cancer has been observed in many families (e.g., Williams, 1978).

- **Interactions**: Enlace a la información de la patología en cuestión:

Interaction	Gene
Affinity Capture-Western	CDKN2A: cyclin-dependent kinase inhibitor 2A (melanoma, p16, inhibits CDK4)
Two-hybrid	CHD3: chromodomain helicase DNA binding protein 3
-	CHD3: chromodomain helicase DNA binding protein 3
Seladin-1 interacts with p53.	DHCR24: 24-dehydrocholesterol reductase
-	NR3C1: nuclear receptor subfamily 3, group C, member 1 (glucocorticoid receptor)
Affinity Capture-MS; in vitro; in vivo	NR3C1: nuclear receptor subfamily 3, group C, member 1 (glucocorticoid receptor)
p53 interacts with GR. This interaction was modelled on a demonstrated interaction between p53 from an unspecified species and GR from an unspecified species.	NR3C1: nuclear receptor subfamily 3, group C, member 1 (glucocorticoid receptor)

En el caso de las interacciones, en el propio resultado se muestra el nombre de la interacción con el gen resultado.

- **Genes**: Enlace a la información del gen en la página del NCBI:

Gene
CDKN2A: cyclin-dependent kinase inhibitor 2A (melanoma, p16, inhibits CDK4)

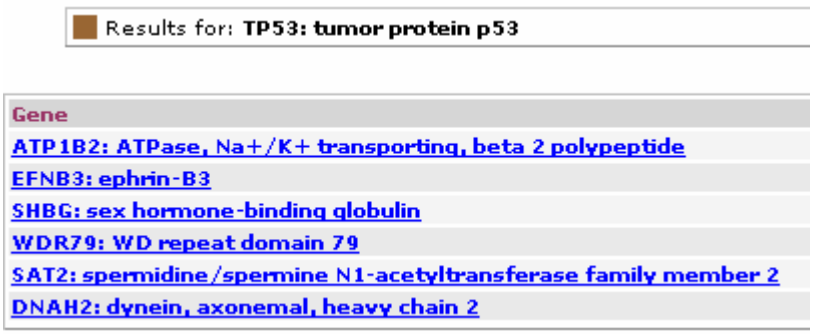
1: CDKN2A cyclin-dependent kinase inhibitor 2A (melanoma, p16, inhibits CDK4) [*Homo sapiens* **]**

GeneID: 1029 updated 25-May-2008

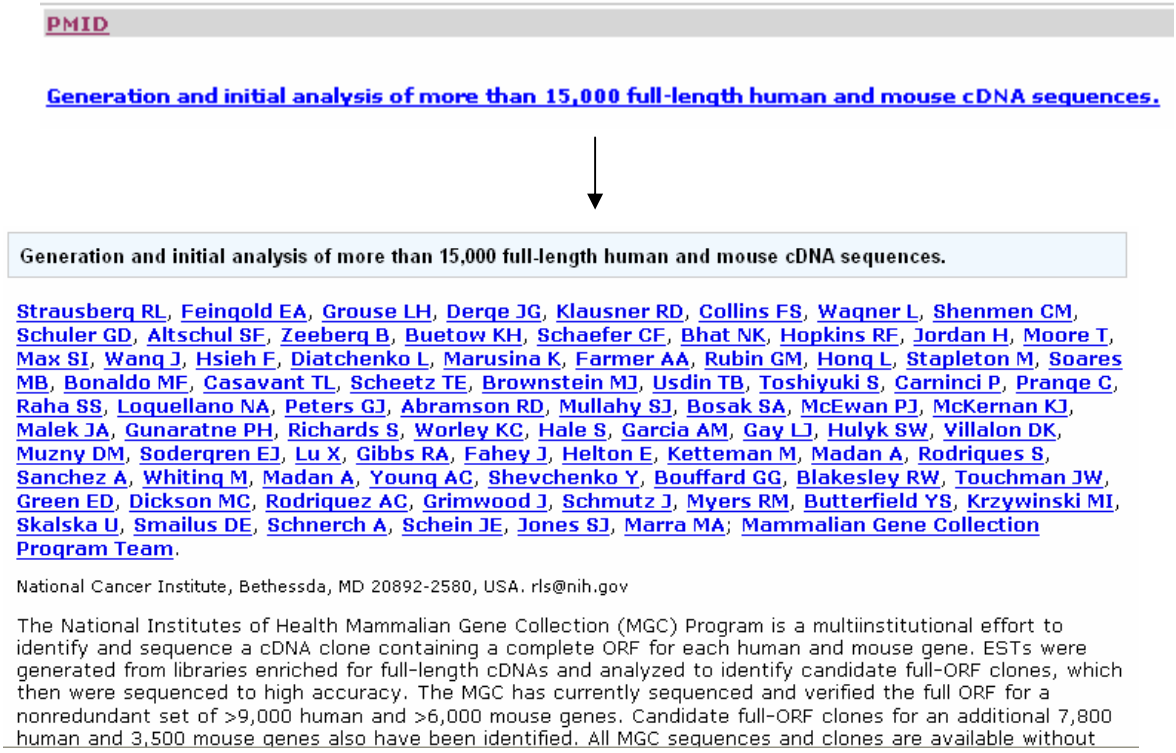
Summary

Official Symbol	CDKN2A provided by HGNC
Official Full Name	cyclin-dependent kinase inhibitor 2A (melanoma, p16, inhibits CDK4) provided by HGNC
Primary source	HGNC:1787
See related	Ensembl:ENSG00000147889; HPRD:02542; MIM:600160
Gene type	protein coding
RefSeq status	REVIEWED
Organism	*Homo sapiens*
Lineage	*Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi; Mammalia; Eutheria; Euarchontoglires; Primates; Haplorrhini; Catarrhini; Hominidae; Homo*
Also known as	ARF; MLM; p14; p16; p19; CMM2; INK4; MTS1; TP16; CDK4I; CDKN2; INK4a; p14ARF; p16INK4; p16INK4a
Summary	This gene generates several transcript variants which differ in their first exons. At least three alternatively spliced variants encoding distinct proteins have been reported, two of which encode structurally related isoforms known to function as inhibitors of CDK4 kinase. The remaining transcript includes an alternate first exon located 20 Kb upstream of the remainder of the gene; this transcript contains an alternate open reading frame (ARF) that specifies a protein which is structurally unrelated to the products of the other variants. This ARF product functions as a stabilizer of the tumor suppressor protein p53 as it can interact with, and sequester, MDM1, a protein responsible for the degradation of p53. In spite of the structural and functional differences, the CDK inhibitor isoforms and the ARF product encoded by this gene, through the regulatory roles of CDK4 and p53 in cell cycle G1 progression, share a common functionality in cell cycle G1 control. This gene is frequently mutated or deleted in a wide variety of tumors, and is known to be an important tumor suppressor gene.

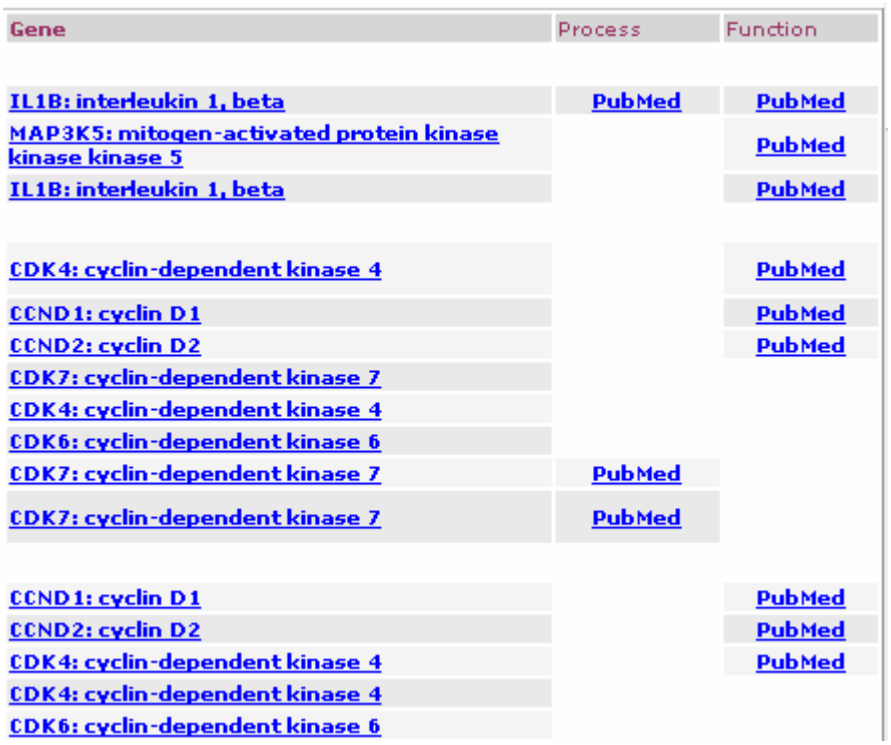
- **MAP**: La operación que busca vecinos de un gen origen no tiene resultado visible en la web de resultado. Tan sólo los genes solución serán aquellos que son vecinos del gen origen:



- **PubMed**: Enlace a la información de la publicación en cuestión:



Por último, destacar que las columnas que se encuentran a la derecha del gen resultado indican las publicaciones PubMed dónde el gen aparece. La cabecera de estas columnas indica el pubmed para la base de datos dónde se ha encontrado un resultado útil para dicho gen.

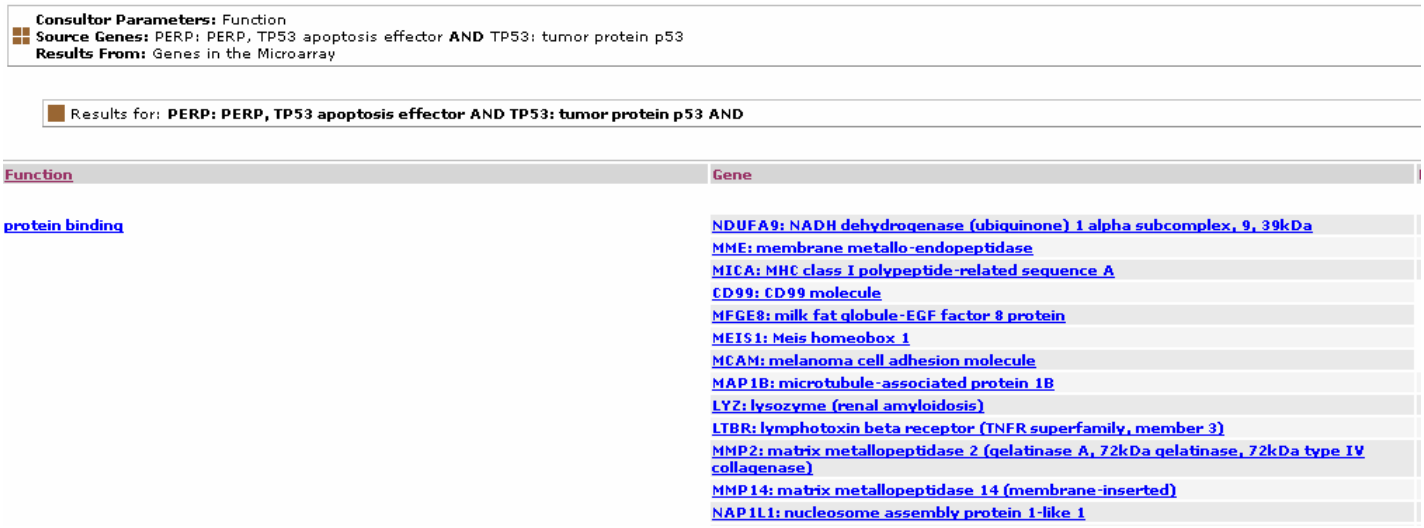


Gene	Process	Function
IL1B: interleukin 1, beta	PubMed	PubMed
MAP3K5: mitogen-activated protein kinase kinase kinase 5		PubMed
IL1B: interleukin 1, beta		PubMed
CDK4: cyclin-dependent kinase 4		PubMed
CCND1: cyclin D1		PubMed
CCND2: cyclin D2		PubMed
CDK7: cyclin-dependent kinase 7		
CDK4: cyclin-dependent kinase 4		
CDK6: cyclin-dependent kinase 6		
CDK7: cyclin-dependent kinase 7	PubMed	
CDK7: cyclin-dependent kinase 7	PubMed	
CCND1: cyclin D1		PubMed
CCND2: cyclin D2		PubMed
CDK4: cyclin-dependent kinase 4		PubMed
CDK4: cyclin-dependent kinase 4		
CDK6: cyclin-dependent kinase 6		

En la imagen observamos los genes resultados de la consulta, y a la derecha, las publicaciones asociadas a los Procesos y Funciones dónde el gen interviene junto con el gen origen. Esta información es un apoyo documentativo más para el usuario que ha realizado la consulta, dotándole de datos que les pueden ser útil en su labor de investigación.

Los huecos vacíos indican que no existe publicación asociada a la base de datos que indica dicha columna.

Por último destacar, que si en lugar de hacer un **OR** con los genes orígenes de la consulta, realizamos un **AND**, los resultados serán aquellos comunes para todos los genes orígenes. Se realiza una intersección de todos los genes resultados de cada gen origen, quedándonos con aquellos comunes en todos.



Consultor Parameters: Function
Source Genes: PERP: PERP, TP53 apoptosis effector **AND** TP53: tumor protein p53
Results From: Genes in the Microarray

Results for: **PERP: PERP, TP53 apoptosis effector AND TP53: tumor protein p53 AND**

Function	Gene
protein binding	NDUFA9: NADH dehydrogenase (ubiquinone) 1 alpha subcomplex, 9, 39kDa
	MME: membrane metallo-endopeptidase
	MICA: MHC class I polypeptide-related sequence A
	CD99: CD99 molecule
	MFGE8: milk fat globule-EGF factor 8 protein
	MEIS1: Meis homeobox 1
	MCAM: melanoma cell adhesion molecule
	MAP1B: microtubule-associated protein 1B
	LYZ: lysozyme (renal amyloidosis)
	LTBR: lymphotoxin beta receptor (TNFR superfamily, member 3)
	MMP2: matrix metallopeptidase 2 (gelatinase A, 72kDa gelatinase, 72kDa type IV collagenase)
	MMP14: matrix metallopeptidase 14 (membrane-inserted)
	NAP1L1: nucleosome assembly protein 1-like 1

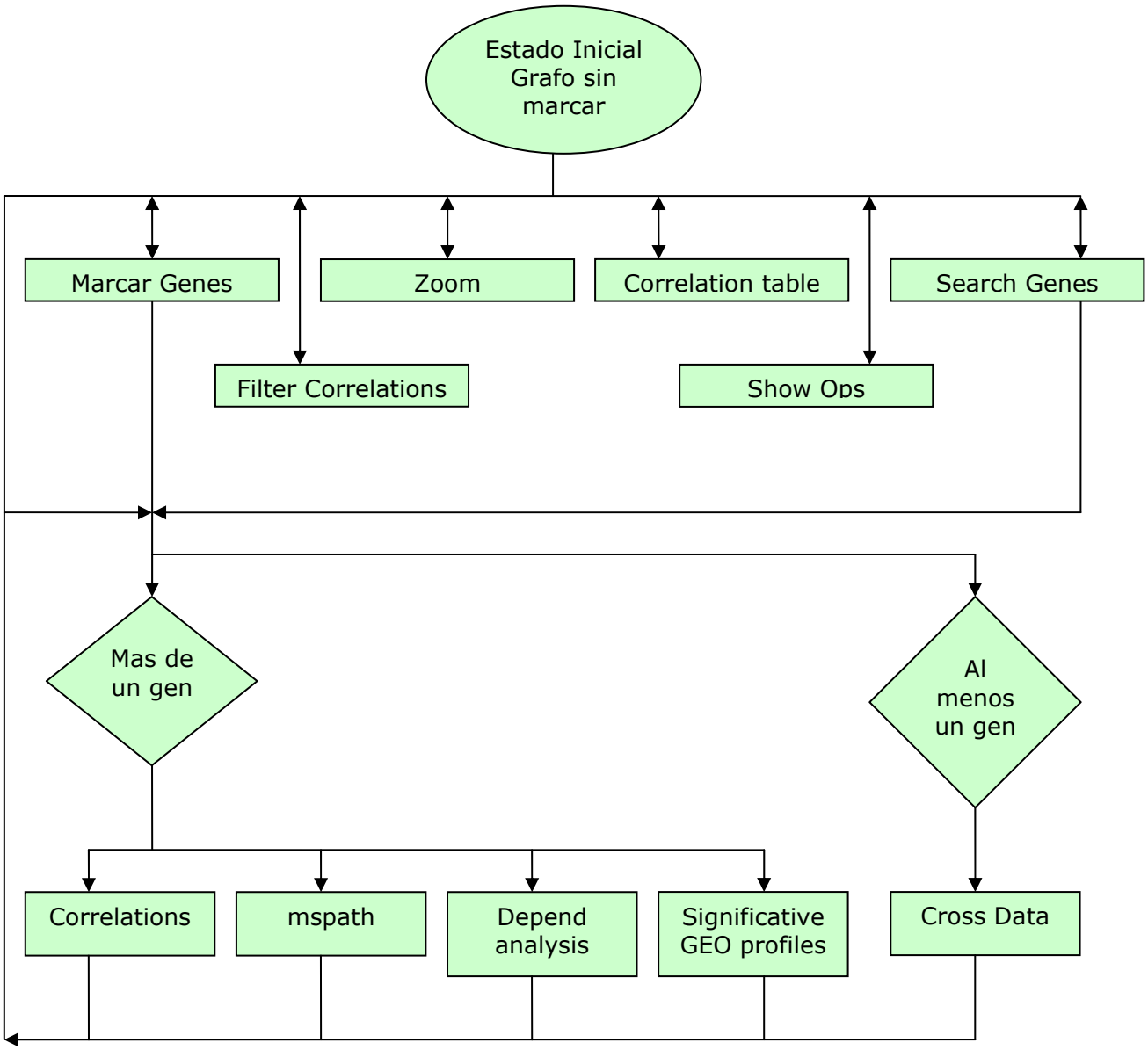
Vemos en esta imagen cómo en la cabecera ya se indica que se realiza un AND para todos los genes origen.

Los resultados mostrados corresponden a los genes comunes obtenidos cómo resultado de realizar una búsqueda de aquellos genes que están en mismas funciones celulares que PERP y que TP53.

5.4.- Mapas de Navegación en la Aplicación:

Los mapas de navegación son una guía gráfica que representa las diversas posibilidades y caminos a seguir para alcanzar la funcionalidad requerida de la aplicación.

Cada nodo representa una funcionalidad de la aplicación, una interacción con el usuario. Partiremos del estado inicial, identificado por el grafo de la microarray sin ningún gen marcado.



En el grafo observamos todas las dependencias entre las diferentes funciones de la aplicación. Así por ejemplo, para poder realizar correlations, o mspath, depend analysis o significative GEO profiles, es necesario que hayan marcados al menos dos genes en la microarray.
Par realizar la aplicación de cruce de información, basta con un gen marcado.

Se representan con líneas los caminos entre una interacción y la siguiente. La flecha indica la dirección de la interacción, de un estado origen a un estado destino.

Este mapa representa pues todas las posibilidades de navegación en la aplicación.

5.5.- Herramientas Usadas en la Elaboración del Proyecto:

Vemos a continuación las características de los sistemas dónde se ha realizado la elaboración del proyecto.
En un primer momento, se empleó un servidor de pruebas dónde ejecutar los robots perl de actualización de datos sobre la base de datos local.
Se descartó este servidor para continuar con la implementación del proyecto debido a su escaso espacio en disco, de tan sólo 1.5 GBytes, siendo los datos a actualizar en la base de datos superiores a dicho tamaño.

- Servidor de pruebas:
    - http://revresearch.phpwebhosting.com
    - Sistema operativo Unix.
    - Base de datos mySQL.
    - Compiladores de Perl y PHP.
    - Conexión FTP para acceso a ficheros remotos.
    - Espacio físico limitado a 1.5 GBytes.

Una vez asegurados que los robots de actualización funcionaban correctamente y trataban la base de datos local sin ningún tipo de fallo ni problema, pasamos a migrarlos al servidor público del departamento de IBB, siendo estas sus características:

- Servidor público:
    - http://revrolutionresearch.uab.es
    - Sistema operativo Unix.
    - Base de datos mySQL.
    - Compiladores de Perl, PHP y Java.
    - Conexión FTP para acceso a ficheros remotos.
    - Espacio físico limitado a 145 GBytes.
    - Óptima conexión a Internet.

La aplicación que hasta entonces funcionaba también en el servidor de pruebas, ha sido migrada por completo al servidor público, con el añadido de la nueva funcionalidad de la operación de cruce de datos. Esta migración será explicada más adelante en el proceso de diseño.

- Robots de actualización de datos:
    - Programados en Perl.
    - Conectan con ficheros de bases de datos externas mediante FTP.
    - Minería de datos sobre los ficheros de datos remotos.
    - Actualizan datos en base de datos local mediante mySQL.

- Applet de cruces de datos:
    - Programados en Java.
    - Integrados en aplicación applet ya existente.
    - Uso de librería gráfica JUNG.
    - Interfaz con cruces de datos mediante llamadas a PHP.

- Aplicación de cruce de datos:
    - Programados en PHP.
    - Hacen de interconexión entre el applet aplicación y la base de datos local y externas.
    - Conexión a base de datos local mediante mySQL.
    - Conexión a base de datos externas mediante eUtils.

- Web de presentación de resultados:
    - Programada en PHP, Html y Javascript.
    - Empleo de hojas de estilo CSS estándar de la página revolutionresearch.uab.es.
    - Se llama como resultado a la aplicación de cruces del applet.
    - Conexión a base de datos local mediante mySQL para disposición de nombres informativos.
    - Conexión a base de datos externas mediante eUtils para disposición de nombres informativos no existentes en la base de datos local.

Aparte de los elementos básicos para construir la completa aplicación de cruces de información mencionados anteriormente, se han empleado una serie de programas que han ayudado a la correcta consecución del proyecto:

- Documentación escrita:
    - Microsoft Word 2003.
    - Microsoft Wordpad 5.1.

- Tratamiento de imágenes:
    - MagicDraw UML 5.5, edición de imágenes UML.
    - Macromedia Fireworks MX, creación de imágenes y botones.
    - Adobe Photoshop 5.5, edición de imágenes.

- Herramientas de soporte para diseño de páginas web:
    - Macromedia Dreamweaver MX.
    - Microsoft Frontpage 2003.

- Herramientas de desarrollo de la aplicación:
    - Apache Server 1.3.23.
    - PHP 4.1.1.
    - Mysql 3.23.48.
    - Perl.
    - Eclipse.
    - Java 1.4.2_07 + Librerías JUNG.
    - Javascript.
    - HTML.
    - Hojas de estilo CSS.
    - Bajo sistema operativo UNIX.

- Herramientas de Internet:
    - Microsoft Internet Explorer 7, navegador web.
    - Firefox 2.0.0.14, navegador web.
    - CuteFTP, programa FTP.

6.- ETAPA DE DISEÑO Y PRODUCCIÓN:

El objetivo del proceso de diseño del sistema de información es la definición de la arquitectura del sistema y el entorno que le dará soporte, junto con la especificación detallada de cada componente.

A partir de esta información se generan las especificaciones de construcción del sistema, así como la descripción técnica del plan de pruebas, definición de requisitos de implantación y diseño de los procedimientos de migración y carga inicial de la aplicación.

Tras finalizar la etapa de planificación y coste, comenzaremos con el diseño de la aplicación. Una vez desarrollado el mapa de la estructura del proyecto en la etapa anterior, continuaremos con el diseño de cada una de las pantallas descritas anteriormente.

6.1.- Diseño de pantallas:

Para cada pantalla de la aplicación he desarrollado inicialmente un pequeño esquema en el que se detallan los principales elementos que formarán parte de ella, así como su distribución y disposición.

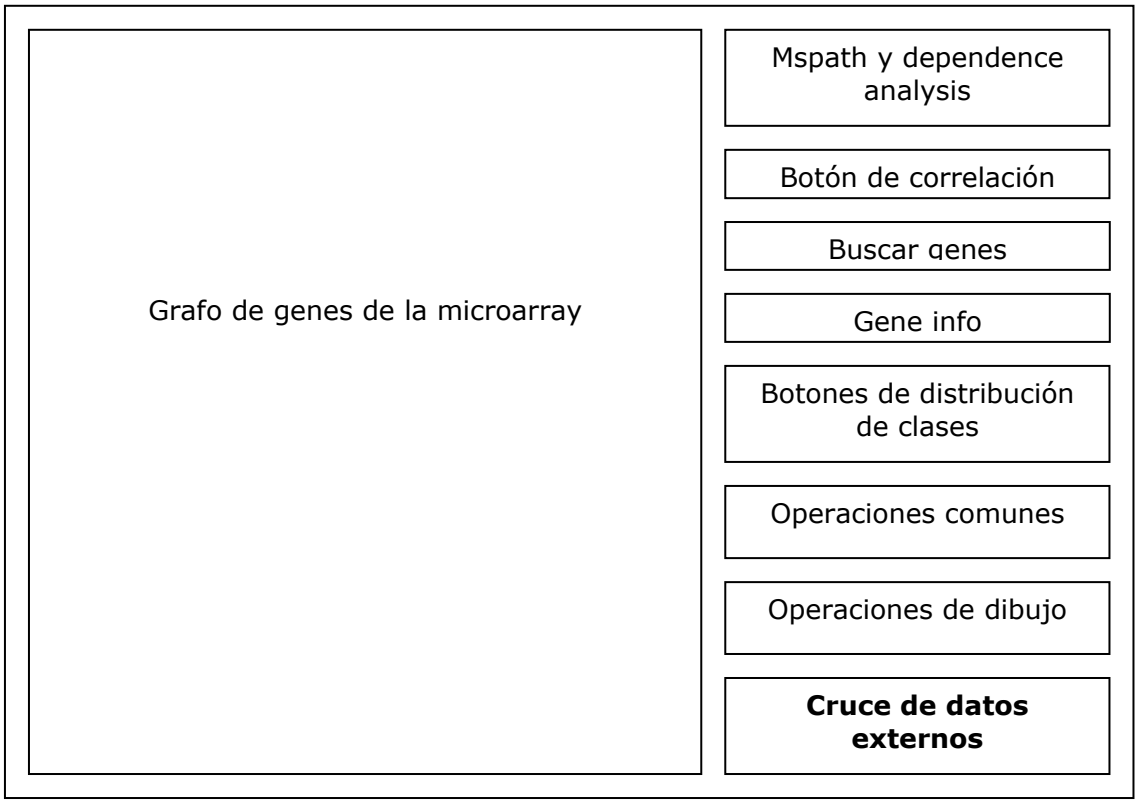
A continuación aparece el resultado final de cada pantalla, dónde se mostrarán los elementos más importantes a la hora de establecer el interfaz con el usuario que trabajará con ellas.

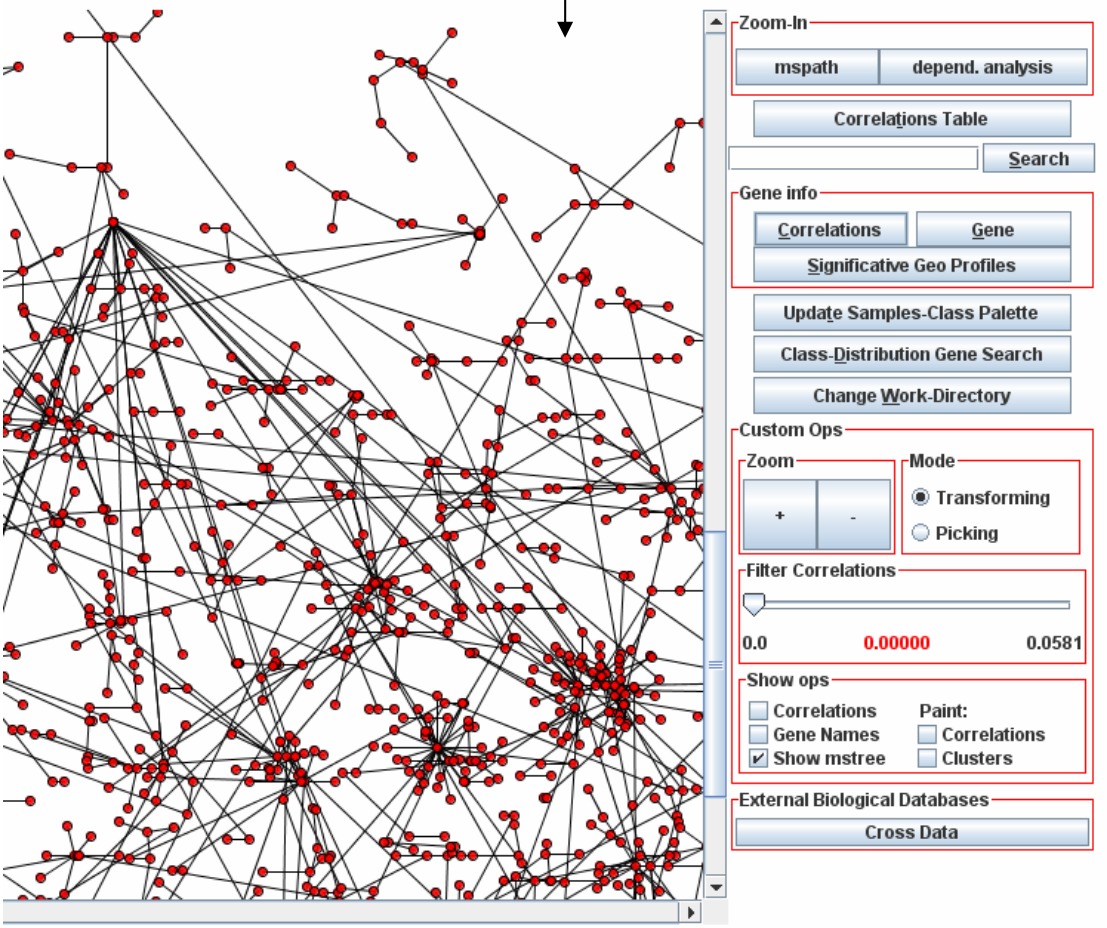
Conforme surgen necesidades durante el proceso de diseño y producción, puede que las pantallas hayan sufrido cambios o modificaciones con la intención de hacer el trabajo de los usuarios más rápido y efectivo.

En la parte de análisis vimos una descripción detallada de las pantallas de la aplicación de cruces de datos. Las pantallas mostradas eran capturas reales de la aplicación final.

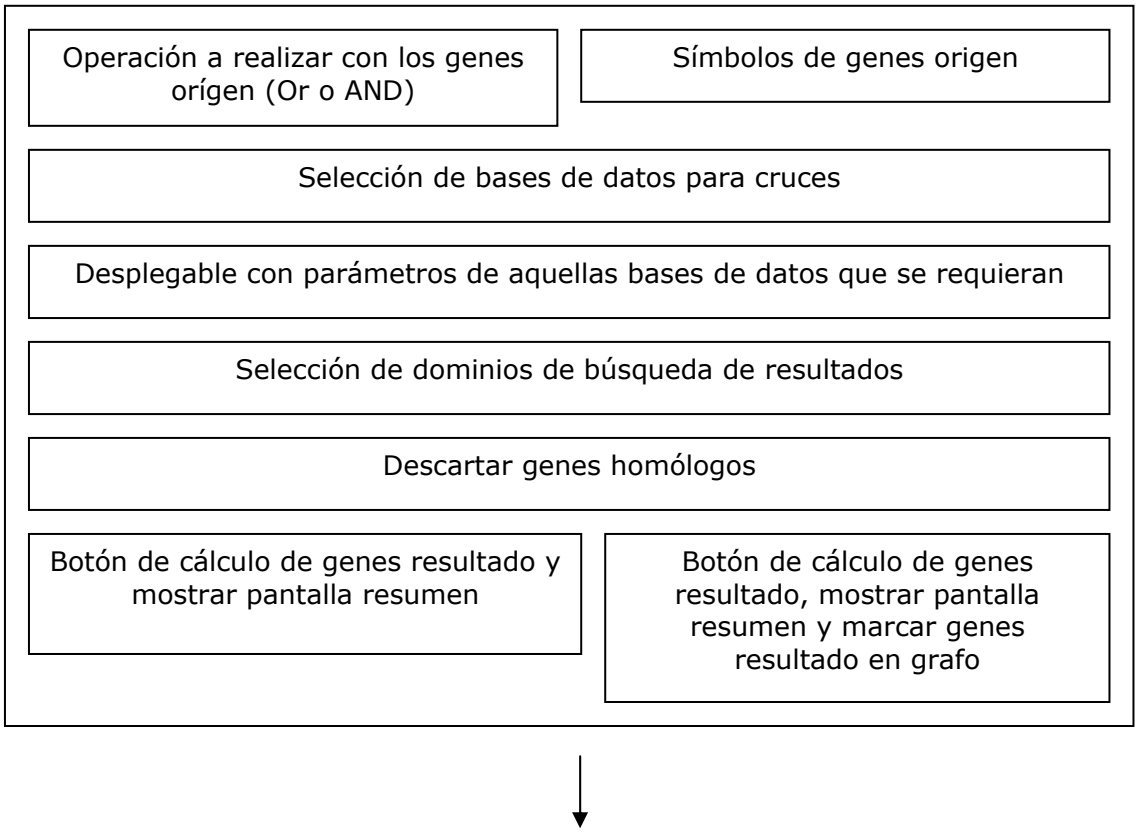
El objetivo de este punto es mostrar el proceso de diseño llevado a cabo a la hora de crear dichas pantallas. Detallar el paso del diseño en papel al diseño digital, empleando para ello las herramientas necesarias mostradas en el punto *5.5 Pag 127*.

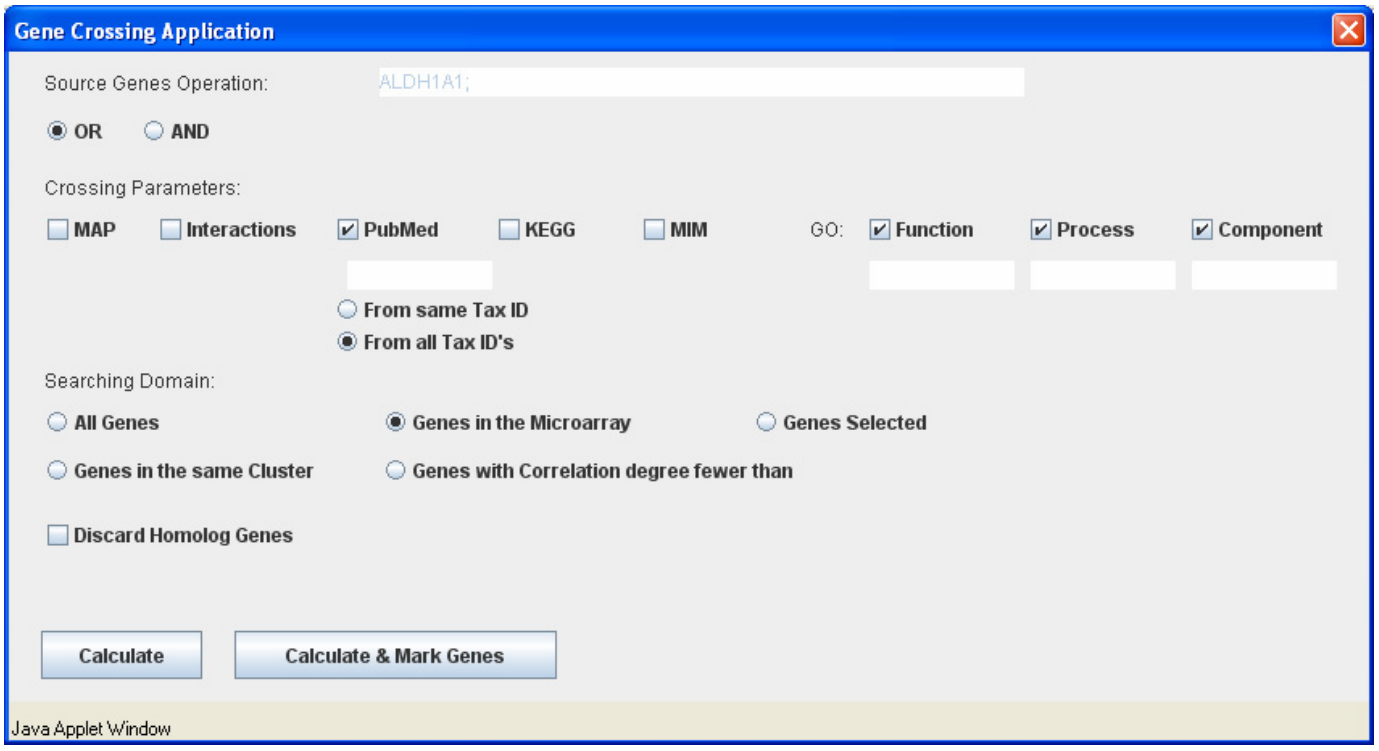
Pantalla del applet aplicación con funcionalidad de cruces de datos remotos:



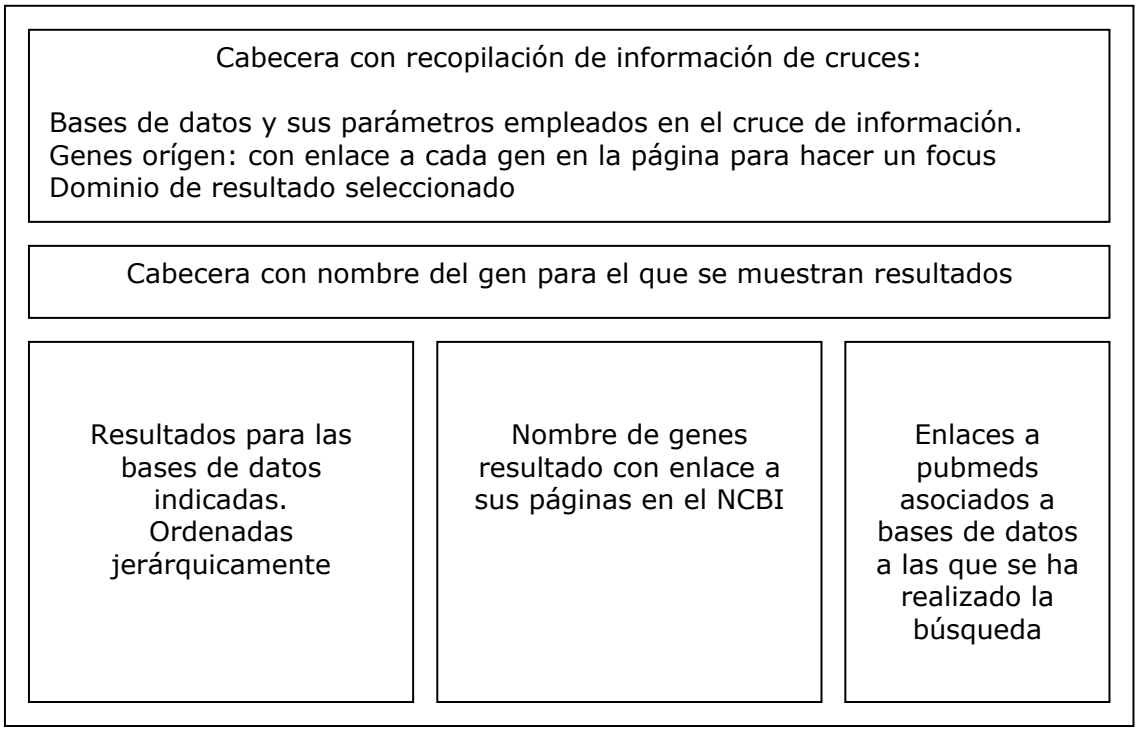


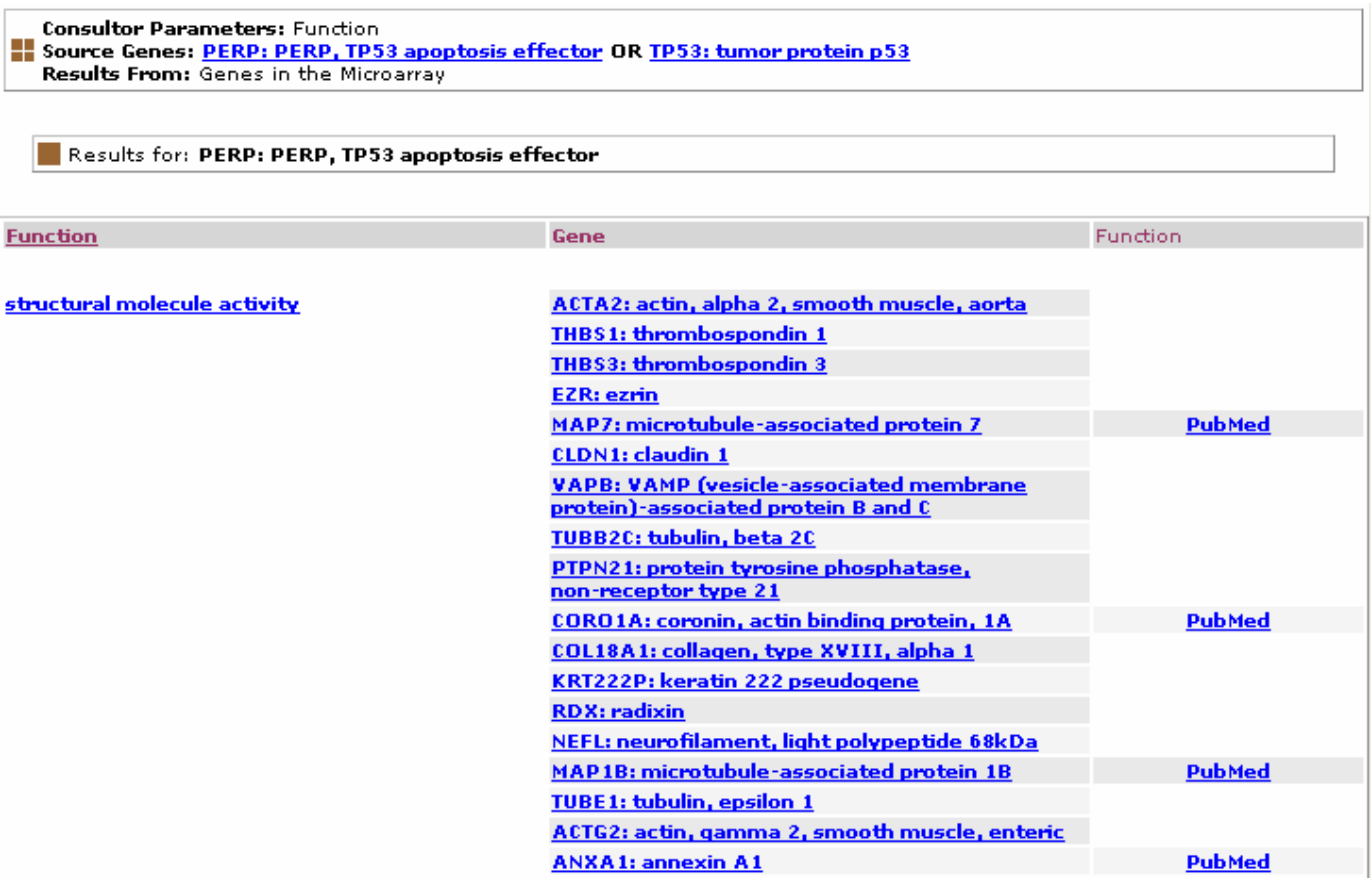
Pantalla de parámetros para realizar cruce remoto de datos:





Pantalla de genes obtenidos cómo resultado de cruces realizados:





Consultor Parameters: Function
Source Genes: PERP: PERP, TP53 apoptosis effector **OR** TP53: tumor protein p53
Results From: Genes in the Microarray

Results for: **PERP: PERP, TP53 apoptosis effector**

Function	Gene	Function
structural molecule activity	ACTA2: actin, alpha 2, smooth muscle, aorta	
	THBS1: thrombospondin 1	
	THBS3: thrombospondin 3	
	EZR: ezrin	
	MAP7: microtubule-associated protein 7	PubMed
	CLDN1: claudin 1	
	VAPB: VAMP (vesicle-associated membrane protein)-associated protein B and C	
	TUBB2C: tubulin, beta 2C	
	PTPN21: protein tyrosine phosphatase, non-receptor type 21	
	CORO1A: coronin, actin binding protein, 1A	PubMed
	COL18A1: collagen, type XVIII, alpha 1	
	KRT222P: keratin 222 pseudogene	
	RDX: radixin	
	NEFL: neurofilament, light polypeptide 68kDa	
	MAP1B: microtubule-associated protein 1B	PubMed
	TUBE1: tubulin, epsilon 1	
	ACTG2: actin, gamma 2, smooth muscle, enteric	
	ANXA1: annexin A1	PubMed

6.2.- Diseño y Programación de Robots de actualización de datos:

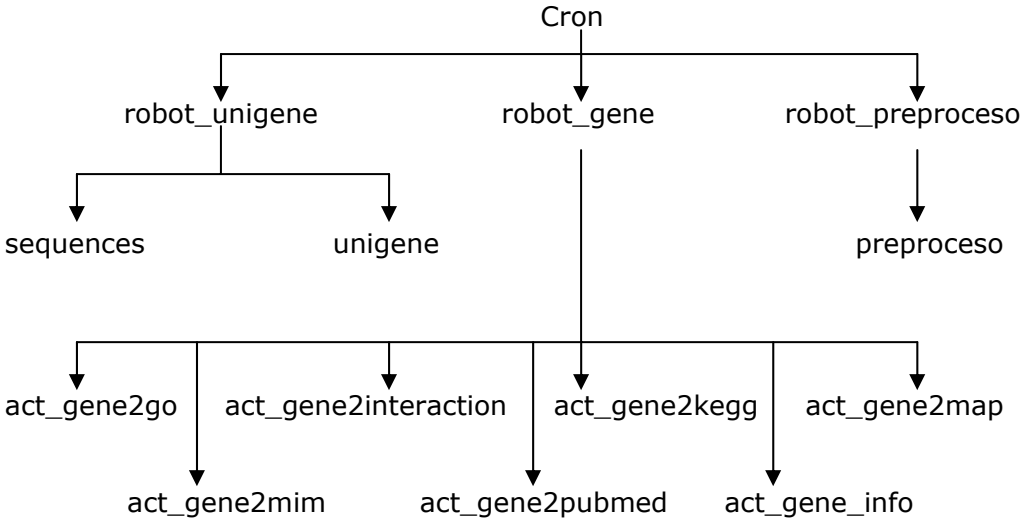
Tendremos 3 tipos de robots programados en perl y situados en el directorio **/var/www/cgi-bin/robots** del servidor público final.

Estos robots serán llamados en orden una vez al mes, según lo programemos en el cron del sistema operativo.
El hecho de emplear esta frecuencia es que se necesita mucho tiempo para actualizar los datos remotos en local. Es necesario bajar ficheros de gran peso y posteriormente obtener la información relevante de su contenido para actualizar la base de datos.

Los tres robots se encargarán independientemente de funciones:

- Actualizar datos relativos a base de datos UniGene, para asociar GeneID's a números de secuencia y códigos UniGene.
- Actualizar datos relativos a base de datos Gene, con toda la información necesaria para realizar los cruces de datos asociados a un GeneID.
- Actualizar nombres de gen de las microarrays existentes.

El árbol de dependencia de los robots de actualización es el siguiente:



Todos los robots bajarán ficheros necesarios del ftp correspondiente, almacenándose estos ficheros en el directorio **/home/robot/unigene/temp**. La estancia en dicho directorio será temporal, pues cuando termine el robot correspondiente de procesar el fichero en cuestión y lo haya actualizado en la base de datos local, lo eliminirá del directorio para no ocupar espacio innecesario.

Veamos el funcionamiento de cada robot por separado:

robot_unigene.pl:

Programa que actualiza las bases de datos locales de Unigene, para obtener los nombres de gen a partir de códigos de secuencia y de códigos Unigene.

Para ello, crearemos una conexión FTP con el sitio UniGene del NCBI. Recorreremos todos los directorios de especies y de cada uno de ellos, tomaremos dos ficheros, el que contiene las secuencias asociada a dicha especie (xx.seq.all) y el que contiene la asociación de Codigo_UniGene con GeneID (xx.data).

Antes de bajar los ficheros, se consultará en la tabla local unigene_ftp la fecha y tamañano del último fichero que se bajó. Si coincide con el actual, no se volverá a bajar ya que los datos estarán actualizados. Si no coincide o no hay entrada en dicha tabla para ese fichero, se procederá a bajar y a insertar los datos en la BD.

Una vez insertados, se actualizará dicha tabla con la fecha y tamaño del fichero bajado.

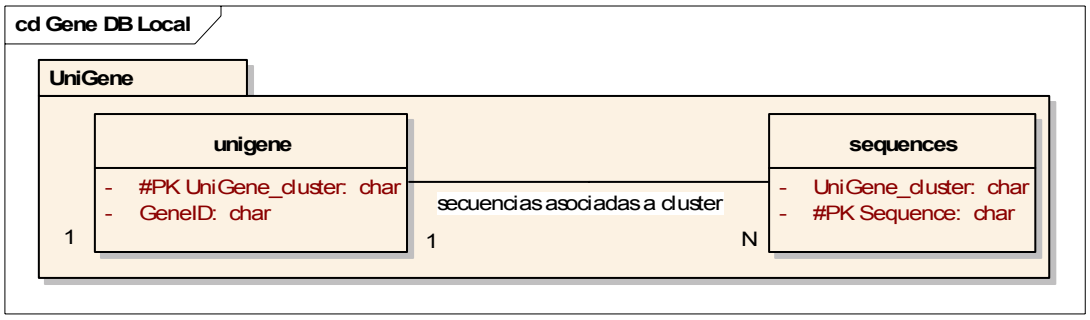
Los scripts perl que insertan los datos bajados son:

perl -X sequences.pl xx.seq.all -> Inserta los pares Secuencia,UniGene_Cluster en la tabla sequences.

perl -X unigene.pl xx.data -> Inserta los datos Unigene (Cod,Title y GeneID) en la tabla unigene.

Los perls sequences.pl y unigene.pl son los encargados de procesar los datos bajados para cada especie de UniGene.

Este robot es el encargado de llamar a los ficheros perl que actualizan datos locales de las tablas:



sequences.pl:

Programa que actualiza la base de datos con las secuencias de UniGene para cada especie.

Para ello, partimos de los ficheros con formato xx.seq.all (xx es la especie) bajados del sitio ftp de Unigene. De este fichero tomamos, para cada identificador Unigene todos los códigos de secuencia que se le pueden asociar.

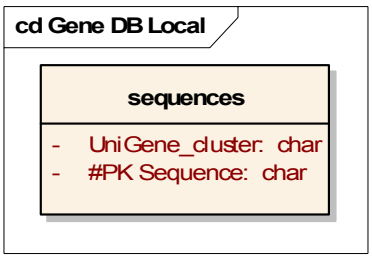
El formato de este fichero .seq.all puede verse en
ftp://ftp.ncbi.nih.gov/repository/UniGene/README

Ejemplo de formato de línea del fichero para cada gen de la especie:

... /clone_end=3' /gb=AA985088 /gi=3163613 /ug=Hs.632256 / ...

Nos quedaremos con el campo GB y UG. El código GB indica la secuencia y el UG el código de UniGene.

Este programa perl pues, a partir de los ficheros .seq.all bajados de cada especie, actualiza los datos de la tabla local:



unigene.pl:

Programa que actualiza la base de datos con los valores de UniGene para cada especie.

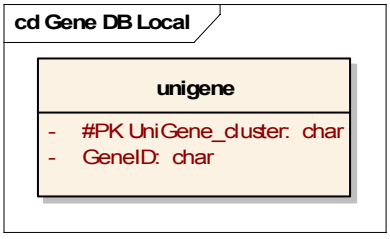
Para ello, partimos de los ficheros con formato xx.data (xx es la especie) bajados del sitio ftp de Unigene. De este fichero tomamos, para cada identificador Unigene el código de Gene asociado (si hay).

El formato de este fichero .data puede verse en
ftp://ftp.ncbi.nih.gov/repository/UniGene/README

Nos quedaremos con el campo ID, GENE_ID.

De esta forma, actualizamos los datos que asocian un código UniGene con un GeneID, pudiendo así trabajar a partir de entonces con códigos GeneID's.

Este programa actualiza la tabla:



robot_gene.pl:

Programa que actualiza las bases de datos locales de gene para obtener todas las características de los genes a partir de su GeneID.

Para ello, crearemos una conexión FTP con el sitio gene del NCBI. Nos bajaremos los ficheros que contienen información necesaria para realizar las consultas cruzadas sobre los genes.

Antes de bajar los ficheros, se consultará en la tabla local gene_ftp la fecha y tamañano del último fichero que se bajó. Si coincide con el actual, no se volverá a bajar ya que los datos estarán actualizados. Si no coincide o no hay entrada en dicha tabla para ese fichero, se procederá a bajar y a insertar los datos en la BD.

Una vez insertados, se actualizará dicha tabla con la fecha y tamaño del fichero bajado.

Los scripts perl que insertan los datos bajados son:

perl -X act_gene2go.pl gene2go -> Inserta la información de ontología de los genes en la tabla gene2go.

perl -X act_gene2interaction.pl geneinteraction -> Inserta los pares de interacciones entre genes en la tabla gene2interaction.

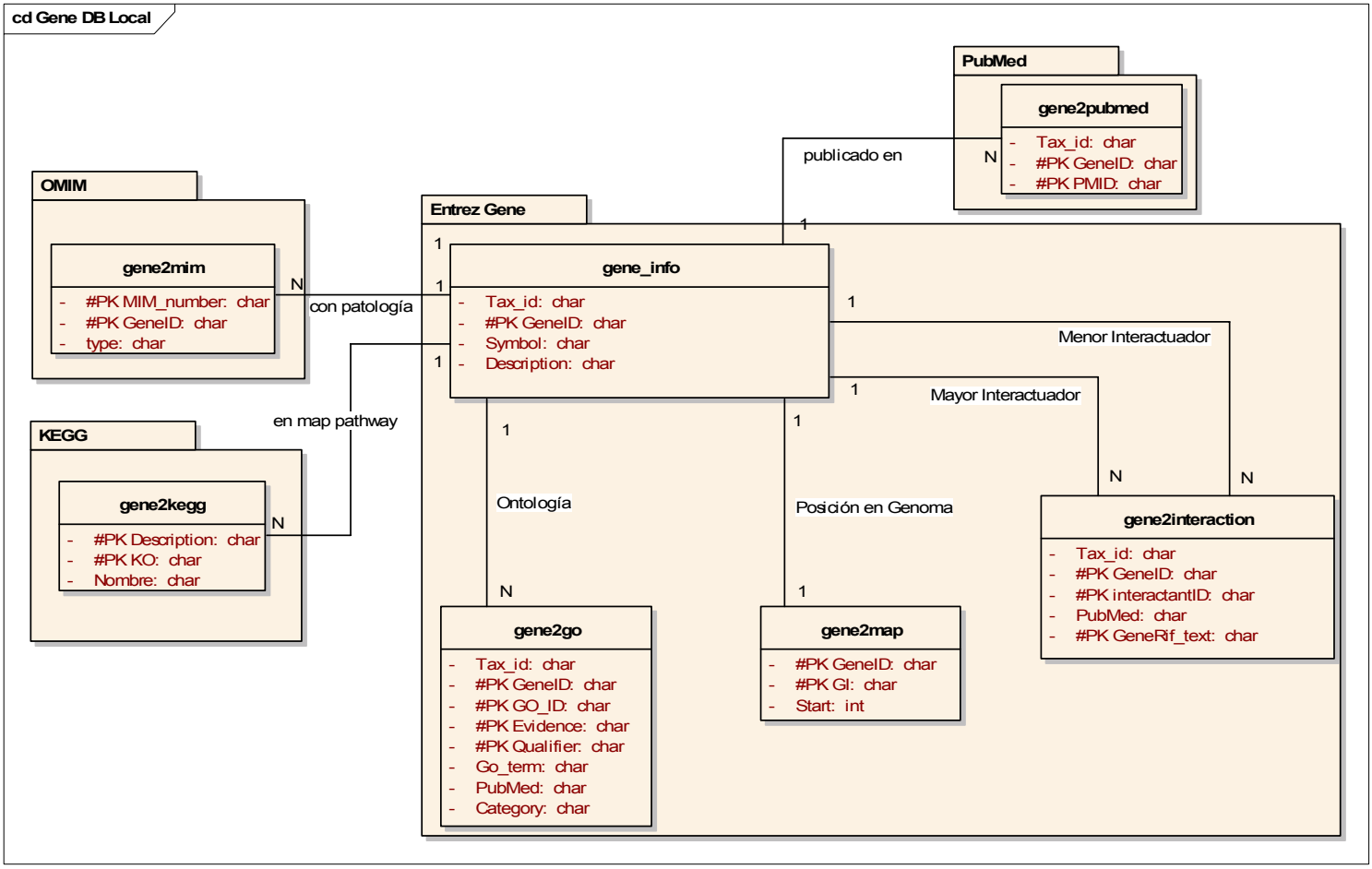
perl -X act_gene2pubmed.pl gene2pubmed -> Inserta la información de pubmed de un gen determinado en la tabla gene2pubmed.

perl -X act_gene_info.pl gene_info -> Inserta la información de los genes, tax_id, GeneID, Symbol y Description.

perl -X act_gene2map.pl gene2refseq -> Inserta la información de posición de los genes en los cromosomas. De esta forma podemos conocer qué genes son vecinos de uno dado.

perl -X act_gene2kegg.pl ko -> Inserta la información de pathways del kegg asociados a un gen en la tabla gene2kegg. El fichero *KO* con esta información será necesario bajarlo del FTP del KEGG y no del NCBI.

Este robot es el encargado de llamar a los ficheros perl que actualizan datos locales de las tablas:

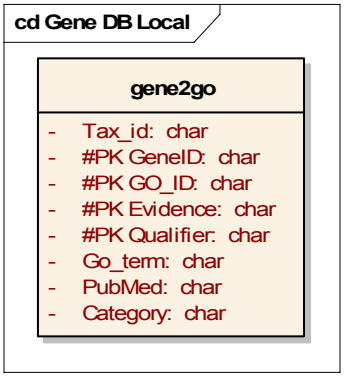


act_gene2go.pl:

Programa que actualiza la base de datos con los valores de ontología para cada gen.

Partiendo del fichero gene2go, extraemos sólo la informacion que nos será util en la BD local.

Nos quedaremos con los campos: Tax_id, GeneID, GO_ID, Evidence, Qualifier, GO_term, PubMed y Category.



act_gene2interaction.pl:

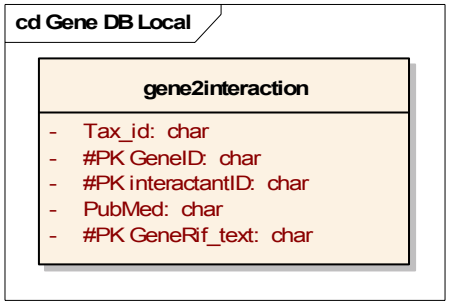
Programa que actualiza la base de datos con los valores de interacciones entre genes.

Partiendo del fichero geneinteraction, extraemos sólo la informacion que nos será util en la BD local.

Nos quedaremos con los campos: Tax_id, GeneID, interactantID, PubMed y GeneRif_text.
El GeneID corresponde al primer interactuante y el codigo del segundo viene dado por interactantID. Si este codigo no es un gen (indicado por "-") no se añadirá la interacción a la base de datos.

Pubmed es una lista con todos los artículos dónde sale la interacción.

Este script actualiza datos en la tabla:

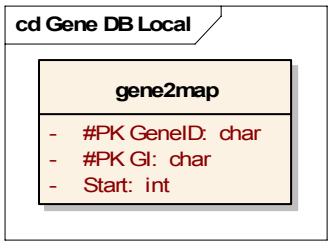


act_gene2map.pl:

Programa que actualiza la base de datos con los valores de posición de un gen en el cromosoma.

Partiendo del fichero gene2refseq, obtenemos la posición dónde empieza el gen a partir de su GeneID. Así podemos conocer los vecinos de dicho gen, aplicando un rango a ambos lados de esta posición dada.

Nos quedaremos con los campos: GeneID, GI y Start:

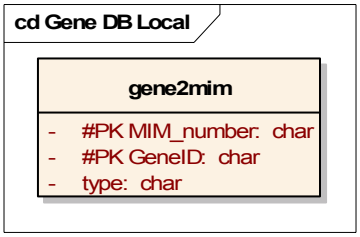


act_gene2mim.pl:

Programa que actualiza la base de datos con los valores de patologías para cada gen.

Partiendo del fichero mim2gene, extraemos sólo la informacion que nos será util en la BD local.

Nos quedaremos con los campos: MIM_number, GeneID y type.

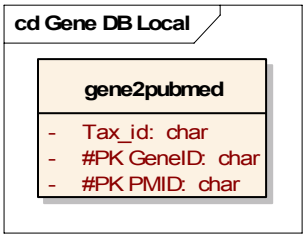


act_gene2pubmed.pl:

Programa que actualiza la base de datos con las publicaciones en las que aparece cada gen.

Partiendo del fichero gene2pubmed, extraemos sólo la informacion que nos será util en BD local.

Nos quedaremos con los campos: Tax_id, GeneID, PMID.

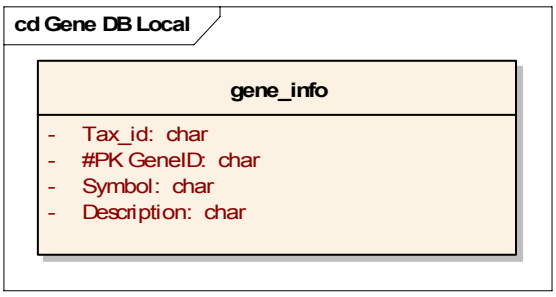


act_gene_info.pl:

Programa que actualiza la base de datos con los valores de Gene para cada especie.

Partiendo del fichero gene_info, extraemos sólo la informacion que nos será util en la BD local.

Nos quedaremos con los campos: Tax_id, GeneID, Symbol y Description.

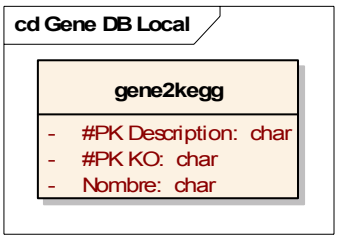


act_gene2kegg.pl:

Programa que actualiza la base de datos con los valores de mapas del KEGG para cada gen.

En este caso se parte del fichero KO obtenido del FTP remoto del KEGG. De este fichero extraemos la información relevante para cada gen.

Nos quedaremos con los campos: Description, KO y Nombre del mapa del KEGG.
Puesto que KEGG no es una base de datos del NCBI, no trabaja con GeneID's. En este caso, para hacer corresponder un gen con sus mapas del KEGG, lo hacemos a partir de su nombre completo.



robot_preproceso.pl:

Programa que recorre los directorios locales de microarrays buscando ficheros de nombres (.genesorig), los cuales actualizará añadiendo en un nuevo fichero .genes los símbolos y nombres de cada gen de la microarray.

Para ello, se recorrerá el directorio donde están los ficheros de microarray: /var/www/cgi-bin/pcop/microarray/

Para cada directorio, se busca el fichero XX.genesorig correspondiente (XX es el número de la microarray). Una vez encontrado dicho fichero se le pasa al script perl *preproceso.pl* que será el encargado de identificar el tipo de fichero de genes a tratar. Los genes del fichero .genesorig pueden estar especificados mediante un código de secuencia, o bien un código UniGene.
Dependiendo del código origen, se consultará a la tabla unigene o a la tabla sequences.

Si el fichero origen dispone de genes con código UniGene, bastará con obtener de la tabla unigene el GeneID asociado a dicho código.
Si por el contrario, el fichero origen trata genes con código de secuencia, habrá que obtener en primera instancia el código UniGene asociado a dicha secuencia de la tabla sequences. Posteriormente, con este código UniGene bastará para obtener el GeneID de la tabla unigene.

La llamada a preproceso.pl para cada microarray es tal y como sigue:

perl -X preproceso.pl /var/www/cgi-bin/pcop/microarray/mXX/XX.genesorig

preproceso.pl:

Programa que a partir de un fichero con códigos de gen, genera uno con los símbolos y nombres asociados a cada gen.

Inicialmente, se le pasará un fichero y tratará de comprobar el tipo de códigos de gen que porta dicho fichero.

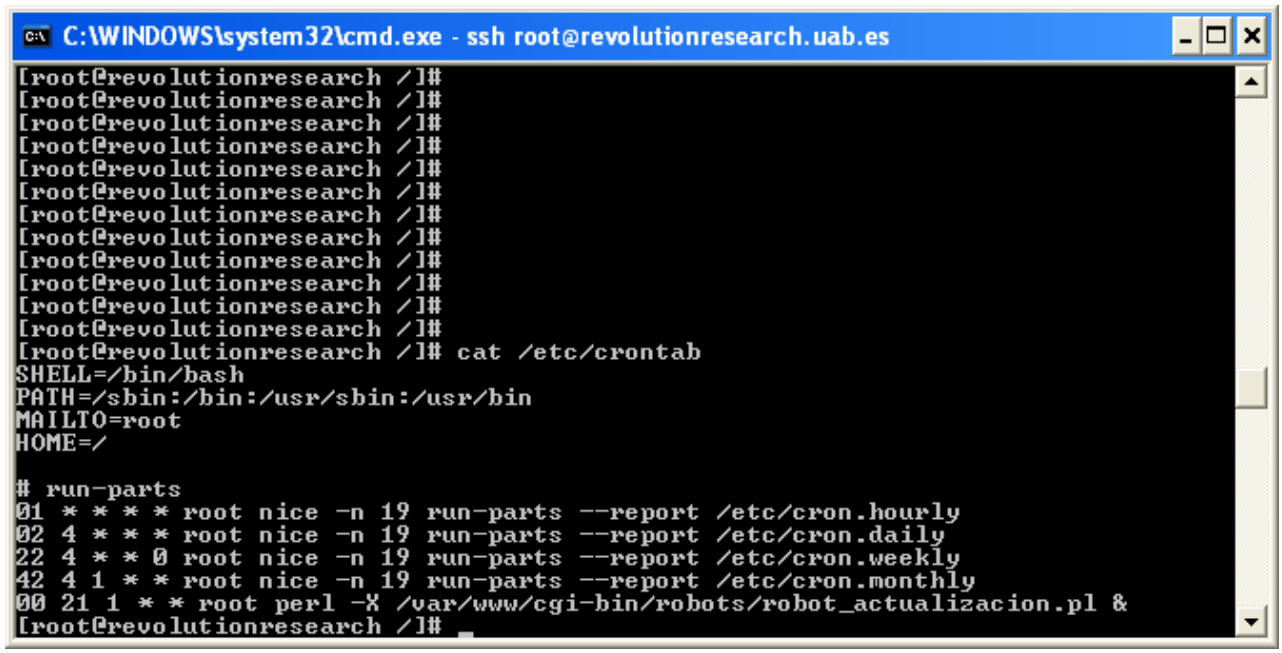
Si no detecta ningún tipo de código de gen, dejará el fichero tal cual.

Si detecta códigos de gen del tipo 3' o 5' (códigos de secuencia), obtendrá el símbolo y nombre del gen a partir del código de secuencia, asociado a un UniGene ID, asociado a su vez a un GeneID del que obtenemos el símbolo y nombre.

Si detecta códigos de gen del tipo Hs.313 (códigos de cluster Unigene), obtendrá el símbolo y nombre del gen a partir del GeneID asociado a dicho código Unigene.

CRON:

Cómo hemos dicho anteriormente, los robots de actualización se ejecutarán una vez al mes. Para ello, será necesario incluirlos en el cron del sistema operativo.
Hemos modificado el fichero /etc/crontab de manera que hemos añadido una nueva línea que contempla la actualización mensual de nuestra base de datos:



En el fichero observamos varios campos:

SHELL es el 'shell' bajo el cual se ejecuta el cron. Si no se especifica, se tomará por defecto el indicado en la línea /etc/passwd correspondiente al usuario que esté ejecutando cron.

PATH contiene o indica la ruta a los directorios en los cuales cron buscará el comando a ejecutar. Este path es distinto al path global del sistema o del usuario.

MAIL TO es a quien se le envía la salida del comando (si es que este tiene alguna salida). Cron enviará un correo a quien se especifique en este variable, es decir, debe ser un usuario válido del sistema o de algún otro sistema. Si no se especifica, entonces cron enviará el correo al usuario propietario del comando que se ejecuta.

HOME es el directorio raíz o principal del comando cron, si no se indica entonces, la raíz será la que se indique en el archivo /etc/passwd correspondiente al usuario que ejecuta cron.

Después de lo anterior vienen las líneas que ejecutan las tareas programadas propiamente. No hay límites de cuantas tareas pueda haber, una por renglón. Los campos (son 7) que forman estas líneas están formados de la siguiente manera:

Minuto Hora DiaDelMes Mes DiaDeLaSemana Usuario Comando

Campo	Descripción
Minuto	Controla el minuto de la hora en que el comando será ejecutado, este valor debe de estar entre 0 y 59.
Hora	Controla la hora en que el comando será ejecutado, se especifica en un formato de 24 horas, los valores deben estar entre 0 y 23, 0 es medianoche.
Día del Mes	Día del mes en que se quiere ejecutar el comando. Por ejemplo se indicaría 20, para ejecutar el comando el día 20 del mes.
Mes	Mes en que el comando se ejecutará, puede ser indicado numéricamente (1-12), o por el nombre del mes en inglés, solo las tres primeras letras.
Día de la semana	Día en la semana en que se ejecutará el comando, puede ser numérico (0-7) o por el nombre del día en inglés, solo las tres primeras letras. (0 y 7 = domingo)
Usuario	Usuario que ejecuta el comando.
Comando	Comando, script o programa que se desea ejecutar. Este campo puede contener múltiples palabras y espacios.

En nuestro caso, ejecutamos el comando:
perl –X /var/www/cgi-bin/robots/robot_actualizacion.pl &

El día 1 de cada mes a las 21:00.

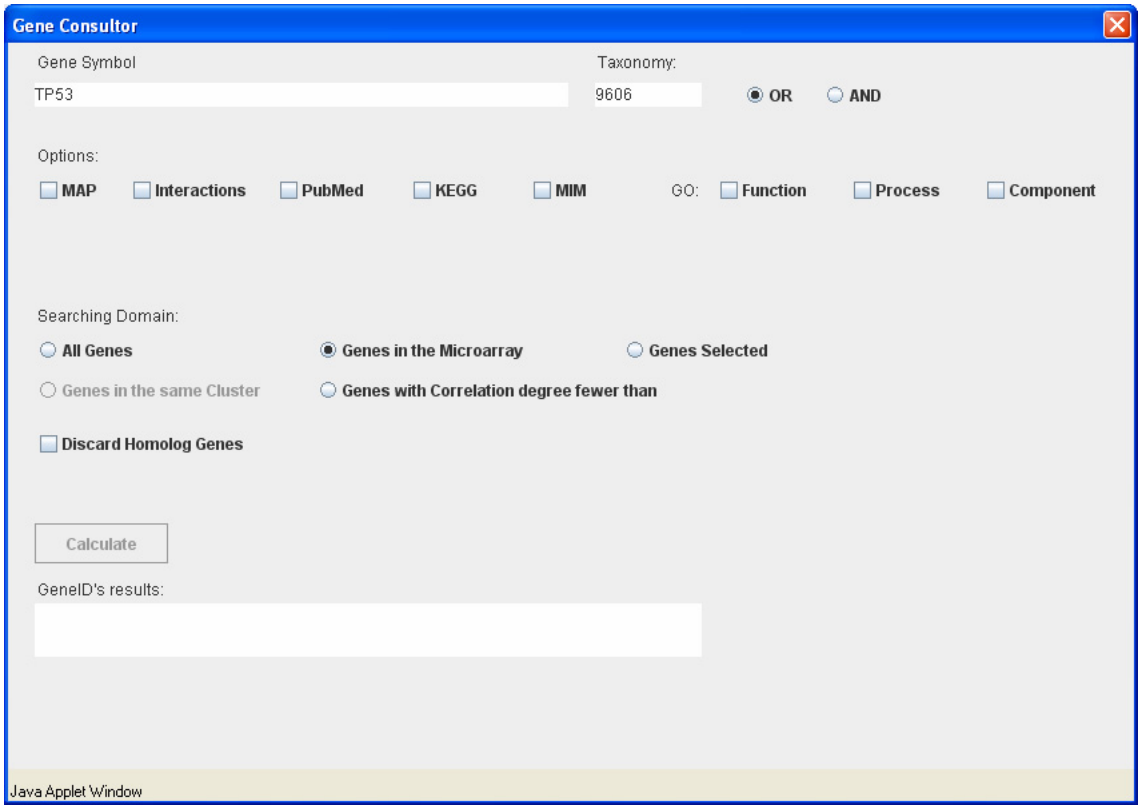
Este fichero robot_actualizacion.pl no es más que un fichero con tres líneas, cada una de ellas llamando a los robots que tenemos programados:

```
$var1=`perl -X robot_unigene.pl 2>$1 &`;
$var2=`perl -X robot_gene.pl 2>$1 &`;
$var3=`perl -X robot_preproceso.pl 2>$1 &`;
```

6.3.- Diseño y Programación de Applet de consultas cruzadas:

En un primer proceso de diseño, creamos un applet independiente del applet aplicación final.
Este applet independiente tenía como objetivo ofrecer toda la funcionalidad respecto los cruces de datos remotos.

Así pues, partiendo de una serie de genes orígenes y taxonomía indicadas por el usuario, este applet generaba la lista de genes resultado del cruce de datos, mostrando por pantalla el listado de genes vía web.



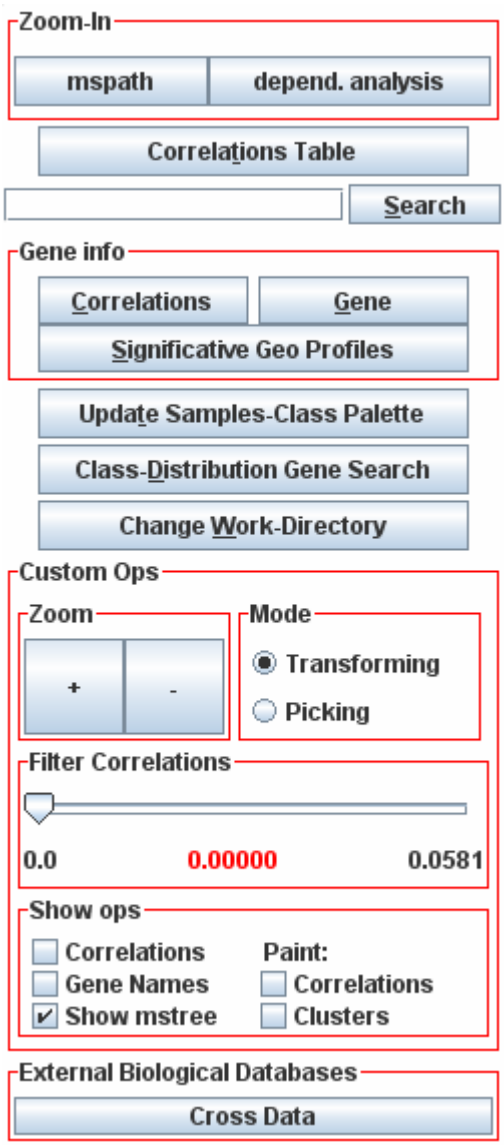
Como vemos, este applet es independiente de cualquier aplicación, funcionando en sí mismo sin necesidad de integración en un entorno.

Para conseguir la total funcionalidad de la aplicación final, tendremos que insertar este applet como parte del applet aplicación ya existente. En este caso, ya el usuario no indicará los genes orígenes como una lista de símbolos, sino que esta entrada vendrá dada por los genes que seleccione o estén marcados en el grafo de la aplicación.
Del mismo modo, tampoco será necesario indicar la taxonomía de la especie, ya que esta vendrá dada por la microarray que se cargue en el grafo de la aplicación.

La salida final, además de la lista de genes resultados completamente detallada vía web, se marcarán los genes solución en el grafo si así lo desea el usuario.

Para la integración del applet de cruces dentro de la aplicación final, ha sido necesario crear un nuevo botón en la aplicación que llame al nuevo applet. Este nuevo applet no será más que una ventana emergente dentro del applet final.

Así pues, en el menú de la derecha de la aplicación tenemos un nuevo botón que realiza el cruce de datos para una serie de genes orígen.

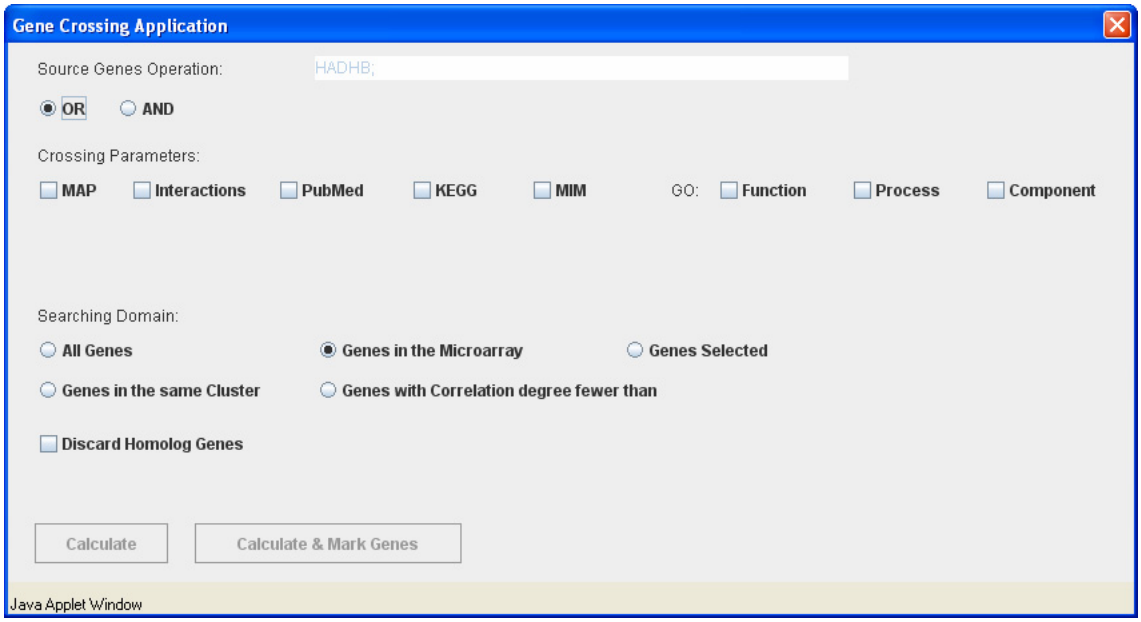


Este botón no hace más que comprobar si al menos hay un gen seleccionado en el grafo. Si así fuera, lanza la ventana emergente para la realización de cruces.

La ventana emergente está construida empleando elementos de la libería Swing de Java (checkboxes, radio buttons, text blocks, buttons …).
Su principal funcionalidad es la de construir de forma correcta la llamada al php que realizará la consulta de cruces de información.

Analicemos, observando la estructura de los campos de la ventana emergente de cruces, cómo construye la llamada para una correcta obtención de genes solución como resultado del cruce de datos elegido por el usuario.

Recordemos que los cruces y consultas a las bases de datos, tanto locales como remotas, la realiza un código php que será documentado más adelante. El applet de cruces es tan sólo un marco de interfaz que construye de forma sencilla y eficiente la consulta que desea realizar el usuario.



El applet emergente de Java almacenará en variables los parámetros indicados por el usuario. Estas variables serán las que posteriormente se pasarán al script de php que realiza las consultas a partir de los datos recibidos:

- **gene**: variable que contiene la lista de símbolos de genes origen separados por ;
- **logic**: variable que contiene la operación lógica a realizar con los genes resultados (OR –resultados para cada gen origen- o AND –resultados para todos los genes origen-).
- **tax**: variable que indica la taxonomía de la microarray.
- **microarray**: variable con el número de microarray que se está mostrando en el grafo.
- **opc**: valor que indica la combinación de bases de datos seleccionadas para el cruce. Cada base de datos tiene asociado un número potencia de 2:

  #Interaction=1
  #MIM=2
  #MAP=4
  #Pubmed=8
  #Function=16
  #Process=32
  #Component=64
  #KEGG=128

  La variable opc contendrá el valor de la suma de las bases de datos seleccionadas. Si por ejemplo vale 34, se habrá seleccionado MIM+Process. Si vale 138, se habrá elegido KEGG+Pubmed+MIM.

- **topicfunc, topicproc, topiccomp**: variables que recogen si se ha introducido algún parámetro de filtrado de búsquedas en las bases de datos de GO.
- **topicpub, use_tax**: variables para indicar parámetro de filtrado a la hora de buscar en pubmed (topicpub) y si se busca en los pubmeds del gen con misma taxonomía que la microarray o en todas las especies dónde aparezca el gen.

- **dom**: variable que indica el dominio de búsqueda de los resultados.

  #1: All genes
  #2: Microarray
  #3: Cluster
  #4: Factors
  #5: Genes origen

- **umbral**: variable que contiene el umbral elegido en caso de buscar en el dominio de factors.
- **use_homo**: variable para indicar si se filtra por homologene los resultados.
- **time**: se usa la variable time para crear un fichero resultado. El fichero de genes resultado a crear tendrá como nombre la hora en la que se hizo la consulta. Con esto se asegura que varias consultas al mismo tiempo no sobreescriban el mismo fichero.

Una vez se tienen todas las variables adecuadas a las opciones que ha marcado el usuario en el interfaz, se construye la consulta y se manda al php encargado de realizarla.

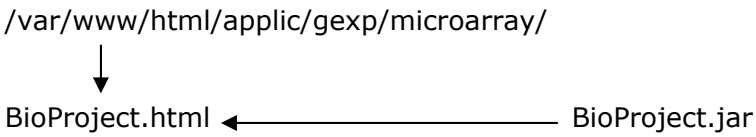
La llamada a la consulta es tal que así:

"http://revolutionresearch.uab.es/applic/gexp/microarray/cruces/cruce.php?opc=" +opc+"&time="+time+"&tax="+tax_id+"&gene="+gene+"&dom="+dom+"&micro array="+id_matrix+"&umbral="+umb+"&topicfunc="+functop+"&topiccomp="+co mptop+"&topicproc="+proctop+"&use_tax="+use_tax+"&topicpub="+topic_pub+" &use_homo="+use_homo+"&logic="+logic_origen;

El php de cruces empleará las variables recibidas que no estén vacías (aquellas que ha marcado el usuario en el interfaz gráfico) las procesará, hará las consultas pertinentes a las bases de datos solicitadas y generará los resultados.

El applet se alojará en el servidor público, en el directorio:
/var/www/html/applic/gexp/microarray/

En dicho directorio encontramos el .jar que contiene todo el entorno gráfico y funcionalidad de la aplicación, así como la página htm que carga dicho applet y el conjunto de todos los ficheros php que utilizan los botones de la aplicación.



6.4.- Diseño y Programación de PHP de cruces y de resultados:

Como hemos visto en el punto anterior, la aplicación de cruces del applet final tiene cómo objetivo generar la cadena de llamada al script php que ejecutará las consultas a las bases de datos.

Este php se denomina cruce.php y se encuentra en el directorio:

/var/www/html/applic/gexp/microarray/cruces/

Además del script *cruce.php*, existe en el directorio el script *mostrar.php* que proporcionará el resultado.

cruce.php:

El fichero de cruces recibe todos los parámetros pasados mediante el entorno gráfico del applet. A partir de estos parámetros, realiza las siguientes acciones:

- A partir del valor **opc** que indica las bases de datos elegidas para hacer la consulta, desglosa su valor en las potencias de 2 que conforman la suma. A partir de este desglose el fichero de cruces ya conoce qué bases de datos participarán en la consulta.

- Dependiendo de si la operación lógica elegida es OR o AND, calculará los genes resultado para cada gen origen por separado (OR) o para todos los genes origen a la vez (AND, genes resultados comunes para todos los genes origen).

- Además desglosa la cadena de símbolos de genes origen separadas por ; y obtiene el GeneID para cada gen (a partir del symbol y del tax_id pasado por parámetro).

- Para cada base de datos seleccionada, se buscan los genes resultado y se van haciendo operaciones **and** anidadas entre consulta y consulta, de manera que cómo resultado final tenemos los genes que cumplen todas las condiciones impuestas. Es decir, aquellos genes que están relacionados con el origen en todas las bases de datos seleccionadas por el usuario. (Ej: si se ha seleccionado la base de datos PubMed, MIM e Interactions, se obtendrán como resultado aquellos genes que estén en mismas publicaciones, en mismas patologías y que interaccionen con el gen origen).

- El script empleará si cabe los parámetros de filtrado para las bases de datos del GO y de PubMed, de manera que se obtendrán genes resultados para las bases de datos según los parámetros introducidos. (Ej: si se ha seleccionado Function con parámetro Protein Binding, sólo se obtendrán como resultados aquellos genes que estén en misma función que gen origen y que esta además sea la función de Protein Binding).

- Se obtendrán genes resultados del dominio seleccionado por el usuario.

    - Para obtener genes de la microarray, se buscan resultados de entre aquellos que aparecen asociados a la microarray en proceso en la tabla de microarrays.

    - Para obtener genes que tengan un grado de correlación menor que uno definido por el usuario, se empleará el parámetro **umbral** indicado en la interfaz. En el directorio de cada microarray existe un

directorio de factors, con ficheros para cada gen de la microarray indicando en cada uno de ellos los genes con los que está relacionado y en qué factor.

- Para obtener genes en mismo cluster que los seleccionados, se emplea el fichero BioFACTORS2K5.net que contiene toda la información acerca de clusters en la microarray.

- Para obtener genes entre los seleccionados, sólo hay que comprobar que entre los genes resultados están los genes origen.

- Para obtener como resultado genes de entre todos los existentes, no hay más que consultar las bases de datos, sin hacer un filtrado posterior de dominio. Los casos anteriores son una especialización de buscar entre todos los genes, ya que se llega hasta el mismo punto, salvo que a continuación se hace un filtrado más, dependiendo del dominio en el que estemos buscando.

- Los resultados obtenidos se almacenarán en un fichero en formato xml. Este fichero tendrá como nombre la fecha de consulta (hora, minutos y segundos) y el nombre del gen origen para el que se han obtenido resultados. Si la operación lógica es un AND, sólo habrá un fichero resultado con todos los genes solución para todos los genes origen.

- Si se ha seleccionado filtrar los genes resultados por homologene, se harán consultas remotas a eutils para cada gen resultado, obteniendo sus homólogos y desechando de entre los resultados aquellos que lo sean.

- Como último paso, el script devuelve al applet los genes obtenidos cómo resultado, de manera que éste puede pintarlos en el grafo de la microarray.

Dependiendo si el usuario ha pinchado en el botón **Calculate** o en el botón **Calculate & Mark Genes** de la interfaz del applet, éste decidirá si los resultados que ha recibido del script php los pintará en el grafo (*calculate & mark genes*) o pasará de ellos y sólo mostrará la web resumen de resultados (*calculate*).

Una vez obtenido los genes resultado, el siguiente paso por parte de la interfaz applet es llamar al script que se encarga de mostrar el resultado ordenado en forma jerárquica y con enlaces de interés e información.

Este script se llama **mostrar.php** y se encuentra en el mismo directorio que el script de cruces:

/var/www/html/applic/gexp/microarray/cruces/

La llamada por parte del applet al script será como sigue:

```
String
show="http://revolutionresearch.uab.es/applic/gexp/microarray/cruces/mostrar.php?opc="+logic_origen+"&gene="+gene+"&time="+time;
```

En este caso necesitamos sólo estos tres parámetros, de manera que el script pueda leer el fichero de resultados de genes.

- Con logic_origen detecta si debe de leer de varios ficheros (opción OR) o de sólo uno con todos los resultados (opción AND).
- La lista de genes es necesaria para separarlos en símbolos y componer el nombre del fichero de resultados para cada gen origen.
- La hora también es necesaria, para acceder, junto con el símbolo del gen, al fichero .xml dónde se encuentran los resultados.

El funcionamiento de este script es sencillo, y se detalla a continuación.

mostrar.php:

- Inicialmente descompone la lista de símbolos de gen en símbolos por separado, para acceder a cada fichero de resultados (en el caso de que la opción fuera OR, si fuera AND sólo existiría un fichero resultado que se identifica con la hora en la que se creó la consulta).

- A continuación, construye la cabecera principal de información, dónde se recogen datos tales cómo las bases de datos empleadas, los nombres de genes orígen, el dominio de búsqueda de genes resultado y si se ha empleado filtro por homologene.

- El siguiente paso es procesar cada fichero .xml de resultados. Para ello se recorren todos los correspondientes a la hora especificada como parámetro.

- En la página de resultados se mostrará una cabecera para cada gen origen, mostrando a continuación los genes que se han obtenido cómo resultado. Estos genes aparecen ordenados jerárquicamente según las bases de datos en las que se han consultado.

- El usuario puede cambiar el orden con sólo pinchar en el nombre de la base de datos que quiere que aparezca en un nivel de la jerarquía superior. *(VER PÁGINA 117)*.

- Todo parámetro mostrado en la página de resultados constará de un enlace a la página oficial del NCBI (o del KEGG), dónde explica en más detalle dicho parámetro.

- Estos parámetros pueden ser genes resultado, lista de pubmeds, mapas del KEGG, definición de ontologías, descripción de interacciones, información del GO.

- Se entiende que el empleo de estos enlaces facilita en gran medida la lectura y comprensión de la lista de resultados por parte del investigador.

- Sólo una lista de genes resultado sería relevante, pero no tan potente y eficaz cómo una lista de genes resultado ordenados jerárquicamente por bases de datos de búsqueda, con enlaces a toda la información que se muestra y que ha sido empleada para el cruce de datos.

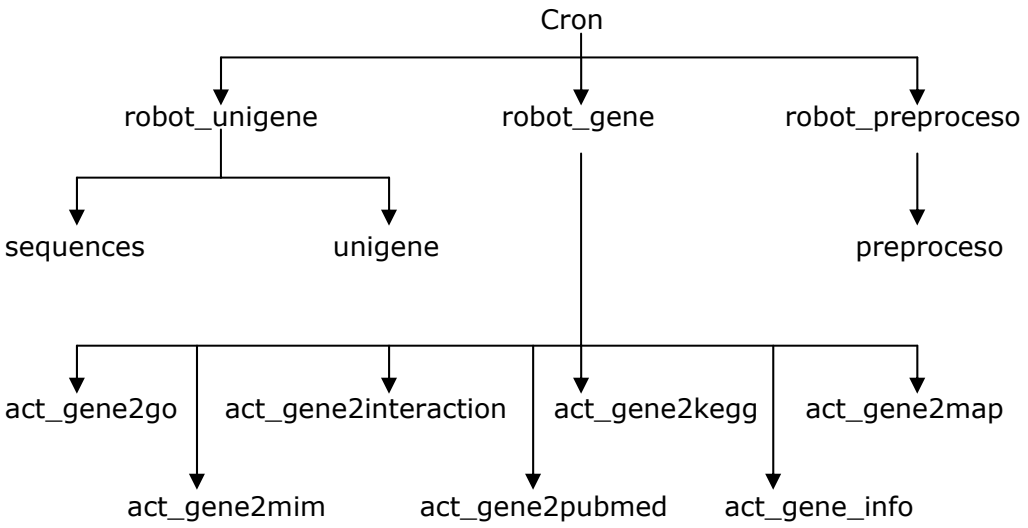
6.5.- Funcionamiento conjunto de la aplicación final:

La aplicación ha sido migrada del servidor de pruebas en el que se encontraba (revresearch.phpwebhosting.com) al servidor público del departamento de IBB (revolutionresearch.uab.es).

Todos los ficheros necesarios para el funcionamiento de la aplicación han sido movidos al servidor de pruebas, con el consiguiente cambio de rutas en los códigos fuente para que todo funcione correctamente.

La aplicación se encuentra situada en diversos directorios del servidor público:

/var/www/cgi-bin/robots/
Robots de actualización de la base de datos local y Robots de preproceso de nombres de las microarrays

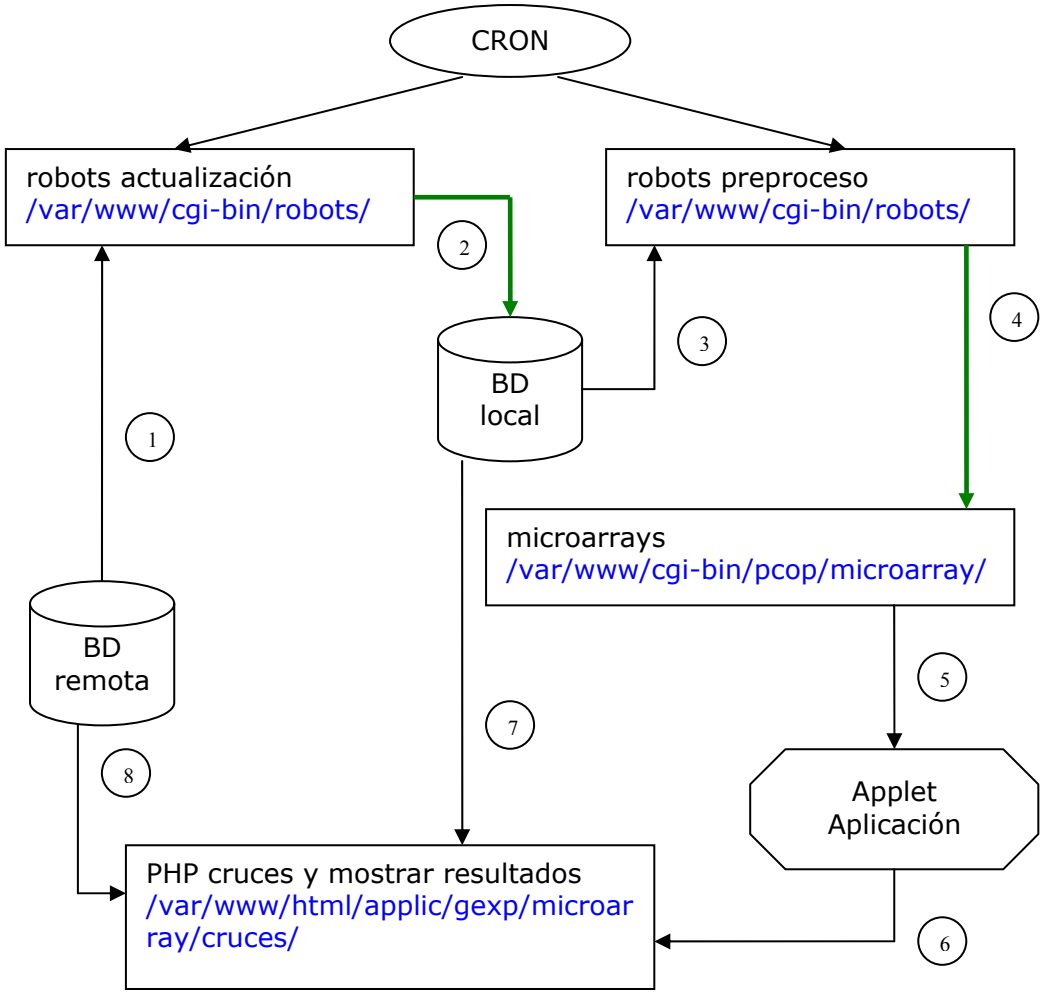


/var/www/cgi-bin/pcop/microarray/
Directorios con ficheros de cada microarray cargada en el sistema.
Cada directorio de la microarray contendrá un directorio llamado factors, en el que aparecen las correlaciones de cada gen de la microarray con el resto de genes.

/var/www/html/applic/gexp/microarray/cruces/
Directorio en el que se encuentran los scripts php de cruce de información y muestra de resultados finales ya descritos en el punto anterior.

/var/www/html/applic/gexp/microarray/
Directorio dónde se encuentra la página web de llamada al applet de cruces:

BioProject.html ◄——— BioProject.jar



En el gráfico observamos el comportamiento completo de la aplicación fuente de este proyecto fin de carrera:

- El cron del sistema operativo ejecuta una vez al mes una serie de robots perl de actualización y de preproceso.
- Los robots de actualización son los encargados de hacer la minería de datos (*pag 47*) de las bases de datos remotas (1), tomando la información necesaria para actualizar la base de datos local (2).
- Los robots de preproceso se encargan de actualizar los nombres de las microarrays existentes (*pag 133*). Para ello, consultan los nombres en la base de datos local (3) que ha sido previamente actualizada y genera los nombres de gen que asigna a las microarrays (4).
- El applet aplicación leerá la microarray que se le pase por parámetro, la pintará completamente en el grafo y dispondrá el menú para que el usuario la manipule a su gusto (5). Además, para las consultas cruzadas de datos remotos, generará la ventana emergente que actúa de interfaz con el usuario. Esta ventana será la encargada de construir la consulta a los ficheros php (6) que realizan la consulta de datos especificada.
- El script php de cruces obtiene la información necesaria de la base de datos local (7) y de la base de datos remota (8) (sólo los datos relativos a la BD Homologene *pag 82*).
- El script php de mostrar resultados obtiene los datos a mostrar de los ficheros que ha generado el script de cruces. Además, para establecer nombres y enlaces a información del NCBI, empleará consultas remotas.

7.- CONCLUSIONES:

Una vez llegamos a esta etapa, el proyecto fin de carrera se encuentra finalizado, y me veo capacitado para exponer las conclusiones a las que he llegado tras completar el proceso de desarrollo.

Hoy día no existe una metodología específica que se aplique sobre proyectos multimedia o proyectos hipermedia que integren la gestión de un entorno web.
Es necesario un estudio de rigor que aporte una metodología eficaz para tratar este tipo de proyectos.

Dada la falta de una metodología estándar, en mi proyecto he seguido el conjunto de fases que se exponen en **METRICA VERSIÓN 3**, para proyectos multimedia enfocados a la web.

Resumen del proceso realizado:

Inicialmente planteamos la realización de las consultas de forma remota, pero debido a las limitaciones impuestas por el propio NCBI, tuvimos que rechazar esta opción. El NCBI sólo permite un número determinado de consultas cada cierto tiempo, y puesto que la aplicación puede ser usada por muchas personas a la vez, esta solución no era viable ya que cortaba la funcionalidad del programa.

Por este motivo, decidimos descargar los datos necesarios para realizar las consultas a la base de datos local. Para llevar a cabo este nuevo objetivo, fue necesario un proceso de investigación exhaustivo de todas las bases de datos que debíamos usar. Era necesario conocer su estructura, campos de cada tabla, datos que contenían, cómo se interconectaban entre ellas… etc.

Una vez que la completa estructura de las bases de datos remotas eran bien conocidas, se planteó realizar un data mining de su contenido. Así pues, construí una serie de robots en lenguaje PERL que descargaban del FTP dónde se almacena la base de datos, sólo aquella información que iba a ser útil en nuestra aplicación de consultas cruzadas de información.

Esta información fue almacenada en primera instancia en un servidor de pruebas, pero debido al gran espacio en disco que ocupaban los datos genéticos necesarios, se optó por implementar la base de datos local en el servidor público del departamento IBB.

Así pues, una vez que la base de datos local estaba bien diseñada y los robots descargaban y la actualizaban de forma correcta, se implementó en el cron del sistema operativo una llamada mensual a dichos robots, de manera que la base de datos queda actualizada cada mes con los nuevos experimentos que se hayan dado de alta en el NCBI.

Posteriormente se implementó el applet en JAVA de la aplicación que cruza la información de genes. Dicho applet se incorporó a la aplicación de gestión gráfica de microarrays ya existente, ampliando su funcionalidad. Además, dicha aplicación fue migrada al servidor público del departamento, de manera que toda la funcionalidad de la misma ya se lleva a cabo desde dicho servidor.

Ha sido necesario pues investigar la documentación de la aplicación applet ya existente, con el objetivo de integrarle la nueva función de cruces de datos.

Una vez todo integrado, el resultado es una aplicación final de tratamiento de microarrays de genes con la posibilidad añadida de cruzar bases de datos remotas.

Objetivo cumplido:

Hemos conseguido integrar en un entorno propio una funcionalidad muy potente. Cualquier usuario investigador puede llegar a conocer, con sólo varios clicks de ratón, información que de otra forma le sería totalmente imposible de alcanzar.

No existía en la actualidad ninguna herramienta que permitiera el cruce de datos genéticos. Si una persona se encontraba estudiando un gen y quería conocer aquellos que aparecieran en la misma patología, o en mismo componente genético (por poner un ejemplo) tenía que investigar manualmente la documentación existente de cada gen para llegar a algún resultado.

Con esta nueva aplicación se consigue un ahorro de tiempo y de trabajo para cualquier investigador. Con tan sólo indicar los parámetros de búsqueda que desean realizar, obtienen una serie de genes resultado de entre todos los existentes. Esta labor sería impensable hacerla manualmente, investigando publicaciones o documentos especializados.

Es un proyecto muy ambicioso, ya que facilita enormemente la tarea de un investigador. La realización de búquedas y cruces de datos en un entorno tan cambiante y tan amplio en número de genes, se ve muy favorecida con esta herramienta. Se consiguen resultados de búsquedas muy complejas y de rango muy amplio en tiempos del orden de los segundos.

En tan sólo unos segundos, se consiguen resultados que de otra forma, serían imposibles o muy tediosos de conseguir.
He conseguido integrar en una base de datos la información necesaria para enlazar todos los genes existentes según sus características, de manera que buscar genes con características comunes sea una tarea sencilla y rápida. Con la inclusión de parámetros en las búsquedas se consigue además filtar la información según los criteros y el enfoque del usuario investigador.

Esta herramienta tan potente puede ser un punto de apoyo muy importante a la hora de acelerar la investigación actual en el campo de la bioinformática y genética.

Además la funcionalidad se ve mejorada con el hecho de que se muestran los resultados ordenados según parámetros seleccionados por el usuario, con enlaces a todo tipo de información que facilite la comprensión de los resultados.
Por si fuera poco, dichos resultados pueden ser pintados en el grafo de la microarray, sirviendo de entrada para una nueva investigación, consiguiéndose así un entorno de trabajo muy potente en el que investigar genes y sus características.

Labor personal:

El trabajo a lo largo del desarrollo del proyecto ha sido muy gratificante y enriquecedor. Ha sido necesario investigar en profundidad conceptos relacionados con la biología, la genética y la investigación celular.

He necesitado documentarme en profundidad de la estructura de las bases de datos americanas, de cómo se interconectan, qué datos mantienen, cada cuanto tiempo se actualizan, cómo acceder a ellas de forma remota… Todo esto con el handicap de que contienen información acerca de genes y todas sus características, que era una rama de conocimiento totalmente desconocida para mí.

Además he tenido que desarrollar todas las labores posibles dentro del proyecto:

- Planificación inicial de tareas a realizar, siguiendo un proceso de **Ingeniería del Software**.
- Investigación de las bases de datos genéticas remotas.
- Creación de **robots** de actualización en **perl** que se ejecutan una vez al mes.
- **Parsing** o procesamiento de ficheros remotos, realizando un **data mining** de su contenido, quedándome con aquellos datos que eran relevantes.
- Conexión remota a bases de datos situadas en norte américa.
- Actualización de la base de datos local mediante conexiones **mySQL**.
- Investigación de applet gráfico de gestión de microarrays ya existente.
- Programación en **JAVA** de un applet para realizar consultas cruzadas.
- Programación en **PHP** de scripts para realizar cruce de datos entre bases de datos remotas y datos locales.
- Generación de ficheros resultados en **XML**, para su posterior representación cómo resultado vía web.
- Conexión de la plataforma JAVA con PHP de manera que los resultados generados por los scripts en PHP son procesados por el applet JAVA y pintados en el entorno gráfico.
- Empleo de librerías **JUNG** de JAVA para dibujar y tratar el entorno gráfico del grafo de la microarray.

Todo ello para conseguir un resultado satisfactorio y sobre todo, con una proyección futura de poder facilitar la labor de investigación genética a usuarios interesados.

8.- BIBLIOGRAFÍA:

- Metodología MÉTRICA Versión 3.

  Ministerio de Administraciones Públicas.

- MÉTRICA Versión 2.1: Metodología de Planificación y Desarrollo de Sistemas de Información. Guías de Referencias.

  Ministerio de Administraciones Públicas.
  Editorial TECNOS, Madrid, 1995.

- Object-Oriented Modeling and Design.

  Rumbaugh,J.
  Prentice-Hall, Englewood Cliffs, New Jersey.

- Ingeniería del Software, un enfoque práctico.

  Pressman, Roger S.
  Mc Graw Hill. 4ª Edición. 1997.

- Análisis y Diseño Estructurado de Aplicaciones Informáticas de Gestión.

  Elena Orta Cuevas. Mercedes Ruiz Carreira.
  Servicio de Publicaciones del Dpto. de Lenguajes y Sistemas Informáticos.
  2ª Edición.

- Gestión y Administración de Bases de Datos.

  Esther Gadeschi Díaz. Juan José Monedero Rojo. Mercedes Ruiz Carreira.
  Servicio de Publicaciones del Dpto. de Lenguajes y Sistemas Informáticos.
  2ª Edición.

BIBLIOGRAFÍA WEB:

- Página oficial del NCBI.

  http://www.ncbi.nlm.nih.gov/
  http://www.ncbi.nlm.nih.gov/sites/entrez/

- FTP del NCBI.

  ftp://ftp.ncbi.nih.gov/repository/UniGene/
  ftp://ftp.ncbi.nih.gov/gene/

- Manejo de herramientas eUtils del NCBI.

  http://eutils.ncbi.nlm.nih.gov/entrez/query/static/eutils_help.html

- Página oficial del KEGG.

  http://www.genome.jp/kegg/

- Conexión a FTP del KEGG.

  http://www.genome.ad.jp/kegg/kegg6.html
  ftp://ftp.genome.jp/pub/kegg/

- Conexiones FTP mediante PERL.

  http://www.xav.com/perl/site/lib/Net/FTP.html
  http://perlenespanol.baboonsoftware.com/tutoriales/modulos/usando_el_modulo_netftp.html

- Conexiones mySQL mediante PERL.

  http://bytes.com/serversidescripting/perl/tutorials/perlandmysql-together/index.html

- Expresiones regulares en PERL.

  http://www.adp-gmbh.ch/cpp/regexp/pcre/index.html

- Programación de applets en JAVA.

  http://bioinformatica.ugr.es/TutorialApplets/tutorialApplets.htm
  http://www.jguru.com/faq/view.jsp?EID=27423
  http://www.realapplets.com/tutorial/GuiExample.html
  http://www.chuidiang.com/java/novatos/JFrame_JDialog.php
  http://java.sun.com/docs/books/tutorial/uiswing/components/internalframe.html

- Manual Oficial de mySQL

  http://www.mysql.com

- Cuadernos de Documentación Multimedia. Modelado de documentación multimedia e hipermedia

  José Manuel Martínez Sánchez. José Ramón Hilera González.
  http://www.ucm.es/info/multidoc/multidoc/revista/cuad6-7/artmulti.htm

- GaiaSur. Consultora Tecnología de la Información.

  http://www.gaiasur.com.ar/

- Laboratorio de Aplicaciones Multimedia.

  http://www.lam-upc.com/

ANEXO 1: EMPLEO DE EUTILS

Durante la implementación del proyeto, ha sido necesario el empleo de estas herramientas para consultas remotas. Si no bien cómo en un principio se esperaba (realizar todas las consultas remotamente) se han empleado para obtener información puntual necesaria para proporcionar los resultados finales.

Así pues, hemos empleado estas herramientas para conocer el nombre asociado a un identificador de gen, a un item del Gene Ontology, nombres de patologías, enlaces a páginas con información, genes homólogos a uno dado, publicaciones pubmeds asociadas a interacciones…

Entrez Programming Utilities

Entrez Programming Utilities son las herramientas que proporcionan acceso a los datos contenidos en la base de datos Entrez y que pueden ser de ayuda para obtener resultados de búquedas en entornos externos al propio NCBI.

Comandos de eutils:

- **EInfo:** Proporciona información de cada campo existente en cada base de datos.
- **ESearch**: Busca y devuelve Identificadores (para usar en Efetch, Elink y Esummary) de genes según los parámetros de búsqueda indicados.
- **EPost:** Crea un fichero con Identificadores para un futuro uso en el entorno del usuario.
- **ESummary**: Devuelve un resumen en un formato especificado por el usuario (txt,htm,xml) de los campos relevantes para una serie de Indicadores indicados cómo parámetros.
- **EFetch**: Devuelve datos en el formato indicado para una serie de Identificadores.
- **ELink**: Busca la existencia de enlaces entre una serie de Identificadores. Devuelve Identificadores y enlaces entre las bases de datos existentes en el NCBI.

Limitaciones en el empleo de eutils:

No sobrecargar el sistema del NCBI. Los usuarios que intenten enviar numerosas peticiones o consultar resultados demasiado complejos serán baneados durante un periodo de tiempo:

- Ejecutar consultas los fines de semana o durante horas de madrugada (hora estadounidense).
- Envíar las consultas a http://eutils.ncbi.nlm.nih.gov en lugar de a la dirección estandar del NCBI.
- No hacer más de una petición cada 3 segundos.

Son estas limitaciones las que nos obligaron en un principio a cambiar la estrategia de realizar todas las consultas remotamente.

Puesto que no podemos permitir que el sistema se vea privado de su funcionalidad debido al contínuo envío de consultas remotas, decidimos descargar en local la información más accesible y sólo emplear las herramientas remotas para obtener nombres e información que mostrar en los resultados finales.

ANEXO 2: EMPLEO DE LIBRERÍAS GRÁFICAS JUNG

Descripción

JUNG(Java Universal Network/Graph Framework) es una biblioteca de software que proporciona un lenguaje común y extensible para modelar, analizar y visualizar los datos que se pueden representar como un grafo o red. Se escribe en Java, que permite que las *JUNG-based applications* hagan uso de las extensas capacidades incorporadas en la API de Java, así como los de otras bibliotecas de terceras personas existentes de Java.

La arquitectura de JUNG se diseña como ayuda para el desarrollo de una gran variedad de representaciones de entidades y sus relaciones, tales como grafos dirigidos y no dirigidos, grafos multimodales, grafos con aristas paralelas e hipergrafos. Proporciona un mecanismo para describir grafos, entidades, y relaciones basados en *metadata*. Esto facilita la creación de herramientas analíticas para conjunto de datos complejos que pueden examinar las relaciones entre estas entidades así como las *metadatas* unidas a cada entidad y relación.

La distribución actual de JUNG incluye la implementación de un gran número de algoritmos de teoría de grafos, minería de datos, y análisis de redes sociales, tal como rutinas de *clustering*, de descomposición, de optimización, generación aleatoria de grafos, análisis estadístico, y el cálculo de distancias en grafos, de flujos, y de otras medidas de importancia (*centrality, PageRank, HITS*, etc.).

JUNG también proporciona un *framework* de visualización que hace más fácil construir herramientas para la exploración interactiva de los datos de la red. Los usuarios pueden utilizar uno de los algoritmos de *layout* proporcionados, o utilizar el propio *framework* para crear sus propios *layouts* personalizados. Además, los mecanismos de filtro que se proporcionan permiten que los usuarios enfoquen su atención, o sus algoritmos, en partes específicas del grafo.

Como biblioteca *open-source*, JUNG proporciona un *framework* común para el análisis y la visualización de grafos y redes.

¿Por qué existe JUNG?

Según la documentación oficial del proyecto JUNG, la creación del *framework* fue una percepción de la necesidad de crear una herramienta genérica, flexible y potente para la creación, manipulación, análisis y visualización de Grafos y Redes en Java. El hecho de que no existieran otras herramientas o APIS para este objetivo, hizo que se trabajara en el desarrollo de una nueva ya que ninguna de las anteriores satisfacía las necesidades y requerimientos en el marco de los proyectos que desarrollaban los autores en sus respectivos departamentos.

Aún así, existen otras herramientas que se pueden adecuar más a las necesidades y capacidades específicas de otros proyectos de menor envergadura.

¿Qué es y qué no es JUNG?

JUNG es un *framework* que permite la creación de herramientas de diseño y visualización de grafos y redes. Puede ser utilizado en el marco de creación de pequeñas porciones de código para probar ideas o en la creación de herramientas con una interfaz gráfica de usuario sofisticada.

JUNG no es en si misma una herramienta final (ni lo intenta ser). Es posible construir una herramienta basada en JUNG, pero para ello es necesario conocer el lenguaje de programación Java. JUNG proporciona pequeños ejemplos para poder desarrollar pequeñas tareas, pero son ejemplos de cómo utilizar JUNG, no son ejemplos de herramientas en el sentido del propio significado de la palabra.

Pasamos a hacer un pequeño resumen de las posibilidades del *framework* donde podremos ver algunas de los objetos usados en este proyecto, no todos, pero si las clases y objetos básicos de los cuales se han derivado los objetos creados para satisfacer los objetivos de éste. La intención de los siguientes apartados es explicar a grandes rasgos las operaciones y las funcionalidades básicas que componen este *framework* para que una persona con unos mínimos conocimientos de Java pueda usarlo.

Grafos, Vértices y Aristas

Propiedades y Operaciones básicas

Grafos, Vértices y Aristas tienen, cada uno, propiedades que se pueden extraer de ellos, y operaciones que pueden realizar ellos (o que se pueden realizar sobre ellos). Las operaciones enumeradas en la parte inferior tienen el comportamiento garantizado para cualquier Grafo, Vértice o Arista que se hayan definido en JUNG. Dependiendo del tipo específico de Grafo, Vértice o Arista, y de la implementación realizada, un objeto como estos, puede tener otras operaciones y características disponibles.

Grafos:

- newInstance(): Retorna un grafo del mismo tipo del grafo que invoca éste método.
- addVertex(v): Añade un nuevo Vértice a este Grafo y devuelve una instancia del mismo.
- addEdge(e): Añade una nueva arista a este Grafo y devuelve una instancia de la misma.
- getVertices(): Devuelve el conjunto de Vértices que tiene el Grafo.
- getEdges(): Devuelve el conjunto de Aristas que tiene el Grafo.
- numVertices(): Retorna el número de Vértices que tiene el Grafo.
- numEdges(): Retorna el número de Aristas que tiene el Grafo.
- removeVertex(v): Elimina el Vértice v del Grafo.
- removeEdge(e): Elimina la Arista e del Grafo.
- removeVertices(s): Elimina todos los Vértices del conjunto s en el Grafo.
- removeEdges(s): Elimina todas las Aristas del conjunto s en el Grafo.
- removeAllVertices(): Elimina todos los Vértices del Grafo.
- removeAllEdges(): Elimina todas las Aristas del Grafo.
- copy(): Realiza una copia del Grafo y de todo su contenido.

Vértices:

- getGraph(): Retorna una referencia del Grafo que contiene este Vértice.
- getNeighbours(): Retorna un conjunto con todos los Vértices conectados por aristas a éste mismo.
- getIncidentEdges(): Retorna un conjunto con todas las Arista incidentes a éste Vértice.
- degree(): Retorna el número de Aristas incidentes a éste Vértice.
- getEquivalentVertex(g): Retorna el Vértice en el Grafo g especificado, si éste, es equivalente al Vértice dado.
- isNeighbor(v): Retorna true si el Vértice especificado v y el Vértice dado, están conectados por, al menos, una Arista, false sino.
- isIncident(e): Retorna true si la Arista e es incidente en el Vértice dado, false sino.
- removeAllIncidentsEdges(): Elimina todas las Aristas del Grafo que son incidentes en este Vértice.
- copy(g): Crea una copia de este Vértice en el Grafo g especificado.

Aristas:

- getGraph(): Retorna una referencia del Grafo que contiene esta Arista.
- getIncidentVertices(): Retorna el conjunto de Vértices que son incidentes a esta Arista.
- getEquivalentEdge(g): Retorna la Arista en el Grafo g especificado, si ésta, es equivalente a la Arista dada.
- numVertices(): Retorna el número de Vértices incidentes en esta Arista.
- isIncident(v): Retorna true si el Vértice v especificado es incidente en esta Arista, false sino.
- copy(): Crea una copia de esta Arista en el Grafo g especificado.

Tipos

El *package* Graph contiene especificaciones (mediante Java interfaces), con varios niveles de abstracción, para grafos, vértices y aristas.

ArchetypeGraph, ArchetypeVertex, y ArchetypeEdge

La interfaz Archetype especifica el comportamiento de grafos generalizados, vértices y aristas; estos se diseñaron para abarcar todo tipo de grafos, incluyendo grafos dirigidos y grafos no dirigidos, grafos con datos incluidos (Ej. aristas con peso), hipergrafos, y grafos con aristas paralelas. Todas las implementaciones de los grafos, vértices y aristas deben implementan una de estas interfaces (o una interfaz que herede de estas interfaces). Los métodos enumerados anteriormente están disponibles para los objetos que implementan dichas interfaces.

Graph, Vertex y Edge

Estas interfaces heredan de las interfaces Archetype y especifican el comportamiento para los Grafos (binarios) en los que cada arista conecta exactamente dos vértices; esta limitación permite la definición de un gran número de nuevos métodos.

DirectedGraph y DirectedEdge

DirectedEdge es un tipo de arista que impone un orden en los vértices incidentes.

DirectedGraph es una interfaz etiquetada para implementaciones de Graph en los cuales su conjunto de aristas son implementaciones de DirectedEdge. Así, por ejemplo, el autor de un método que se crea para operar sólo con

grafos dirigidos debe asegurar que el argumento del grafo pasado al método debe ser un DirectedGraph.

UndirectedGraph y UndirectedEdge
: Estas interfaces son análogas a las correspondientes DirectedGraph y DirectedEdge mencionadas anteriormente, pero para grafos no dirigidos y aristas no dirigidas (donde no se impone un orden en los vértices incidentes).

El *package* graph.impl contiene varias implementaciones de estas especificaciones del *package* Graph. Actualmente, todo el código de este *package* está diseñado para soportar grafos binarios.

AbstractSparseGraph, AbstractSparseVertex y AbstractSparseEdge
: Estas clases son el esqueleto de las implementaciones de las interfaces Graph, Vertex y Edge. Este intento es para minimizar el esfuerzo requerido para implementar dichas interfaces, mientras no sea necesario saber si el grafo es o no dirigido.

  Estas clases, y todas las heredadas a partir de ella, están diseñadas para grafos donde, sólo en algunas ocasiones, el número de aristas es superior al de vértices. Esta no es la mejor implementación para manipular grafos densos, donde cada vértice está conectado a varios vértices.

  Al ser clases abstractas, el usuario no puede crear instancias de ellas.

DirectedSparseGraph, DirectedSparseVertex y DirectedSparseEdge
: Estas clases extienden las clases abstractas de Graph, Vertex y Edge para grafos dirigidos; las clases Graph y Edge implementan DirectedGraph y DirectedEdge interfaces, respectivamente.

UndirectedSparseGraph, UndirectedSparseVertex y UndirectedSparseEdge
: Estas clases extienden las clases abstractas de Graph, Vertex y Edge para grafos no dirigidos; las clases Graph y Edge implementan UndirectedGraph y UndirectedEdge interfaces, respectivamente.

Creando y Añadiendo Elementos

Crear un grafo es directo: llamar al constructor del tipo de grafo deseado.

```
Graph g = new DirectedSparseGraph();
```

Crea un nuevo grafo dirigido y lo asigna a la variable del tipo Graph. (Mirar la Javadoc para los detalles del tipo Graph y de la implmentación DirectedSparseGraph)

También puede crear un grafo leyendo éste desde un fichero (comentado posteriormente en la sección Entrada y Salida) o generándo un *random graph* (comentado posteriormente en la sección Algoritmos).

Una vez haya creado el grafo, puede crear vértices y añadirlos al grafo:

```
Vertex v1 = (Vertex) g.addVertex(newDirectedSparseVertex());
Vertex v2 = (Vertex) g.addVertex(newDirectedSparseVertex());
```

y una vez tiene vértices, puede conectarlos con aristas:

```
DirectedEdge e = (DirectedEdge) g.addEdge(new
DirectedSparseEdge(v1,v2));
```

Se puede observar que se crean vértices y aristas y se agregan al grafo combinando dos operaciones en sólo una línea de código. La naturaleza de estas dos etapas del proceso permiten crear vértices o aristas "huérfanos" y que no formen parte de ningún grafo. Esto se da como compromiso entre las *common practices* en las APIs de Java y los efectos secundarios de constructores y la semántica de grafos. Sin embargo, el comportamiento de un método JUNG para aristas o vértices, con la excepción del método getGraph(), no está especificad para vértices/aristas huérfanos. Las implementaciones de JUNG nunca crearán este tipo de elementos, y se recomienda que los usuarios sigan está práctica jerarquizando la llamada al constructor dentro del método que agrega ese elemento al grafo (como en los ejemplos mostrados).

Algunas restricciones a tener en cuenta:

- Un vértice/arista sólo puede formar parte de un grafo al mismo tiempo.
- Un vértice/arista sólo puede ser añadido a un grafo al mismo tiempo.
- Una arista no puede ser creada y ser incidente en un vértice huérfano.
- Una arista no puede ser creada uniendo vértices incidentes en diferentes grafos.
- El vértice creado debe de coincidir con el tipo de grafo donde se añade. (Así, por ejemplo, no se puede añadir un DirectedSparseVertex en una implementación UndirectedGraph).
- La arista creada debe coincidir con el tipo de vértice que conecta, y con el tipo de grafo en el cual es añadida.

Si algunas de estas restricciones no se cumple, se genera un error en tiempo de ejecución, y se genera una excepción del tipo FatalException . Todas estas restricciones son detectadas en cuanto ocurren, pero puede ocurrir que algunas de estas no sean tan evidentes y el error que se genere sea más sutil y por tanto más difícil de detectar.