UNIVERSIDAD POLITÉCNICA DE VALENCIA

Ingeniería Informática

Desarrollo de una aplicación para la extracción y gestión de información biológica procedente de bases de datos públicas

Proyecto Final de Carrera

Presentada por: Víctor Hernández Estévez

Dirigida por: Ana Martínez Pastor
Pedro Fernández de Córdoba Castellá

Valencia, julio 2011

AGRADECIMIENTOS

Quisiera dar las gracias a todas las personas que han formado parte de mi vida apoyándome en mi formación como ingeniero y en especial a mi familia y a mi novia, que en todo momento me han dado el apoyo que he necesitado. Sin ellos no hubiera sido posible.

Tampoco sin olvidarme de Pedro por darme la oportunidad de conocer en mayor profundidad el campo de la bioinformática y a Ana por apoyarme en todo momento, dando fruto a un proyecto que espero ayude en futuras investigaciones.

Resumen

En este proyecto final de carrera se pretende facilitar el acceso y la gestión a bases de datos públicas a través de una aplicación informática, con el fin de investigar sobre las diversas reacciones químicas que se dan dentro de un conjunto de microorganismos para potenciar ciertas propiedades de los mismos.

Palabras Clave: *Bioinformática, XML, Parser, KEGG*

ÍNDICE

Introducción

La bioinformática, según una de sus definiciones más sencillas, es la aplicación de tecnología de computadores a la gestión y análisis de datos biológicos. Los términos bioinformática, biología computacional y en ocasiones, biocomputación, utilizados en muchas situaciones como sinónimos, hacen referencia a campos de estudios interdisciplinarios muy vinculados, que requieren el uso o el desarrollo de diferentes técnicas que incluyen informática, matemática aplicada, estadística, ciencias de la computación, inteligencia artificial, química y bioquímica para solucionar problemas, analizar datos, o simular sistemas o mecanismos, todos ellos de índole biológica, y usualmente (pero no de forma exclusiva) en el nivel molecular. El núcleo principal de estas técnicas se encuentra en la utilización de recursos computacionales para solucionar o investigar problemas sobre escalas de tal magnitud que sobrepasan el discernimiento humano. La investigación en biología computacional se solapa a menudo con la Biología de Sistemas.

La Biología de Sistemas es el campo de investigación interdisciplinaria de los procesos biológicos en el que las interacciones de los elementos, internos y externos, que influyen en el desarrollo del proceso se representan con un sistema matemático. Este enfoque "global" permite comprender íntegramente el funcionamiento de los sistemas biológicos y profundizar en el entendimiento de cómo sus interacciones internas y con otros sistemas conllevan a la aparición de nuevas propiedades. Prácticamente cualquier proceso biológico puede ser objeto de estudio de la biología sistémica, como por ejemplo, el crecimiento de una célula, la interacción entre dos bacterias o la circulación sanguínea en un organismo.

Como una evolución conceptual de la Biología de Sistemas surge la Biología Sintética (BS) cuyo objetivo principal es el diseño y construcción de sistemas biológicos. La expresión Biología Sintética ha estado presente en la literatura científica y técnica ya en referencias de principios de la primera década del siglo XX. Como tal, el término BS se está convirtiendo en un concepto cada vez más comprehensivo, que:

1. Abarca nuevos marcos teóricos que se ocupan de los sistemas biológicos con las herramientas conceptuales y el lenguaje descriptivo de la Ingeniería.
2. Utiliza enfoques de diseño de organismos con propiedades novedosas inspirados en el diseño de circuitos eléctricos y mecánicos.
3. Persigue la creación de nuevos organismos con propiedades “a la carta” basada en la combinación racional de partes biológicas disociadas de su contexto natural.

En la actualidad el término Biología Sintética (BS) se ha convertido en un concepto que abarca a la totalidad de las investigaciones desarrolladas en la frontera entre la Biología Molecular y la Ingeniería en sentido estricto. El avance de esta disciplina científica se cimentará en el establecimiento de un marco computacional y conceptual que dé asistencia en el desarrollo de sistemas biológicos artificiales modulares basándose en una metodología ingenieril y sistemática, para lo que se necesita proveer a la próxima generación de diseñadores en Biología Sintética y a los futuros biotecnólogos e ingenieros biológicos de nuevas herramientas computacionales integradas en un entorno común para el análisis de fenotipos metabólicos y el diseño de nuevos circuitos genéticos complejos.

Todo esto hace de esta nueva incipiente rama de la ciencia un área eminentemente multidisciplinar que requiere de la colaboración de científicos de diversos campos de conocimiento como biología, química, informática e ingeniería con el objetivo final de poder llegar en un futuro a ser capaces de implementar nuevas funciones en sistemas vivos de forma dirigida.

El desarrollo de software aplicado a la Biología de Sistemas ha experimentado una gran evolución en los últimos años que va desde el desarrollo de un lenguaje estándar para la expresión de modelos como es el SBML (del inglés, Systems Biology Markup Language), hasta el desarrollo de distintas aplicaciones que permitan analizar los sistemas biológicos. En este campo, desde el punto de vista de la construcción de modelos de sistemas globales, habría que resaltar el trabajo que permite automatizar en gran medida la construcción de una red metabólica a partir de la anotación del genoma proveniente de las bases de datos públicas y que en cuyo ámbito se enmarca este proyecto.

En los últimos años ha habido un incremento exponencial de la información que se posee sobre los organismos vivos: genética, procesos de regulación y metabolismo. Dicha información va a parar a grandes bases de datos, varias de ellas disponibles en internet, lo que constituye una valiosa y extensa información en manos de los investigadores sin embargo, aparece aquí una de las mayores preocupaciones. Ésta es debida a que la información biológica que se encuentra en estas diversas bases de datos a distintos niveles (proteómicas, genómicas, transcriptómicas, etc.) de libre acceso vía web como KEGG (Kyoto Encyclopedia of Genes and Genomes) contienen gran cantidad de información de referencia sobre sistemas biológicos que presentan determinadas deficiencias desde el punto de vista de acceso y gestión de dicha información de un modo automatizado. No satisfacen las necesidades de los usuarios debido a la no completitud de la misma , la falta de una nomenclatura unívoca para referirse a los distintos elementos de dichas bases de datos así como la existencia de información no suficientemente contrastada, que hace que el desarrollo del software que tenga acceso a ella, requiera siempre de un proceso de depurado por parte del usuario.

Esto afecta indudablemente a la ejecución de uno de los objetivos de este proyecto, que consiste en la reconstrucción a escala genómica de una red metabólica. Actualmente es un proceso no automatizado e iterativo de toma de decisiones que fácilmente requiere del trabajo de al menos una persona/año para agrupar satisfactoriamente la lista de reacciones metabólicas para un organismo específico.

Por todas estas causas, este proyecto pretende abarcar la parte de la reconstrucción de una red metabólica en lo que a extracción de información genómica sobre bases de datos se refiere. Además se creará un programa encargado de gestionar la base de datos y una interfaz de usuario que facilite la interacción con la misma.

Capítulo 1
BASES DE DATOS PÚBLICAS

En esta sección se pretende comentar las diversas bases de datos públicas que podemos encontrar a través de internet que nos puedan proporcionar datos relacionados con la bioinformática, cuya clasificación viene dada por el campo concreto en el que se especializan. Por ello hay que enumerar esta clasificación y ver con ejemplos qué tipo de datos nos pueden proporcionar cada una de ellas.

Bases de datos de genomas

Se encargan de mantener y actualizar las secuencias y las anotaciones de genomas completos:

- Ensembl (EBI)
- Genome viewer (NCBI)
- Goldenpath (UCSC)

Existen también recursos genómicos especializados:

- Transfact: sitios de unión a factores de transcripción.
- EST: Expressed Sequence Tags.
- UTRDB: Untranslated regions.
- SpliceSitesDB: Pares de señales de splicing.

Bases de datos de proteínas

Secuencias primarias de aminoácidos sin revisión humana:

- Trembl (EBI)
- nr (NCBI)

Con revisión humana:

- Swisprot (EBI)

Bases de datos de proteomas:

- Proteome analysis (EBI)

Estructuras secundarias o dominios. Varían según la fuente de las proteínas y el análisis que se realiza sobre ellas:

- BLOCKS: Motivos alineados de PROSITE/PRINTS.
- PROSITE: Expresiones regulares sobre Swiss-prot.
- PRINTS: Conjunto de motivos que definen una familia sobre Swiss-prot/TrEMBL.
- PFAM: Modelos de Markov sobre Swiss-prot.
- INTERPRO: Integra la información de muchas bases de datos de dominios.

Estructuras tridimensionales de macromoléculas con las coordenadas en el espacio de cada átomo:

- PDB: Base de datos principal de estructuras tridimensionales.
- CATH: Clasificación de PDB en diferentes grupos funcionales y estructurales.
- MMDB: subset de PDB mantenido por NCBI.
- MSD: subset de PDB mantenido por EBI.

Bases de datos de nucleótidos

Las bases de datos de ácidos nucleicos reciben las secuencias de los laboratorios experimentales y las organizan haciéndolas accesibles a diario a toda la comunidad científica. Existen varias bases de datos que intercambian diariamente su contenido.

- Genbank(NCBI)
- EMBL (EBI)
- KEGG (Genome net)

Como hemos podido observar hay muchísimas posibilidades a la hora de escoger una base de datos u otra, por lo tanto hay que tener muy claro el motivo de estudio para hacer la mejor elección y por lo tanto empezar a investigar los modos de acceder a su información.

En el caso de estudio de este proyecto, se va a tratar de simplificar lo máximo posible el entendimiento a nivel biológico, pero sí que es necesario tener claros ciertos términos sin los que no es posible explicar ciertas decisiones en su desarrollo.

El primer término con el que debemos de empezar a familiarizarnos es “pathway” o ruta metabólica., que es una sucesión de reacciones químicas que conducen de un sustrato inicial a uno o varios productos finales, a través de una serie de metabolitos intermediarios. Este término va a ser la base del proyecto, por lo que la elección de la base de datos va a ser la que mayor descripción nos de los pathway.

Al volver a observar las diferentes clasificaciones, vemos que muchas de las bases de datos citadas aparecen en varias de ellas. Esto significa que son bases de datos con fines multidisciplinares y que, por lo tanto, almacenan y procesan información de diversa índole.

En nuestro caso este hecho no es relevante, ya que incluso nos beneficia poder contrastar la información y poder hacer descartes o incluso rellenar datos incompletos. Por lo que la elección se debe basar en la que a priori nos vaya a dar el vasto de la información que necesitemos. De todas las bases de datos disponibles, la que más se acerca a nuestros intereses es KEGG.

Esta base de datos nos proporciona una gran cantidad de información relativa a los pathway sobre aproximadamente unos 1400 organismos. Además de proporcionar diversos métodos para acceder a su información y hacer extracciones. Es cierto que del apartado de bases de datos sobre nucleótidos también tenemos EMBL y Genbank, pero ambas pertenecientes a bases de datos multidisciplinares como comentábamos antes y por lo tanto menos especializadas.

KEGG: acceso a datos

KEGG nos proporciona diversos métodos para conectarnos a su base de datos y por lo tanto acceder de manera pública a su información.

Entre estos métodos, nos encontramos con KEGG SOAP, KEGG REST y KEGG FTP.

Los 3 métodos, al fin y al cabo acceden a la información que se les solicita. Pero hay que tener en cuenta de qué manera accede y cómo nos devuelve la información cada uno de ellos.

KEGG SOAP es un protocolo WebService que conecta a la base de datos y mediante etiquetas nos permite hacer peticiones y por lo tanto incluirlo en nuestro programa. El problema que acarrea este método es la lentitud con la que se hacen estas consultas y tipo de datos (string) que nos devuelve, ya que nos obliga a partir la información para poder construir nuestra propia base de datos; por lo tanto queda descartada.

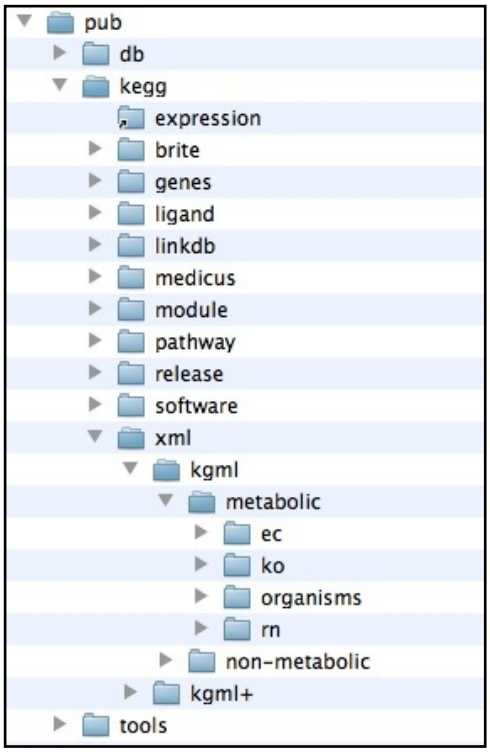
KEGG REST es una interfaz por la que se accede a través de URLs, que en un principio no nos permite extraer información si no es a través de los propios identificadores de KEGG y que además está en vías de desarrollo. Es decir, otro método que no nos es viable.

KEGG FTP es la opción que, coloquialmente hablando, parece más “bruta”, nos proporciona una serie de ficheros XML que nos muestra la estructura organizada por pathways de organismos, reacciones y compuestos. Esta es al fin y al cabo, es la información crucial que necesitamos para empezar a construir nuestra base de datos.

KEGG: sistemas de ficheros

Entender cómo almacena los XML y cómo los estructura KEGG, nos dará una idea de como hacemos el parseo de los mismos para extraer la información.

En primer lugar debemos tener claro la estructura de árbol que muestra su FTP:



Como podemos observar, la dirección de acceso al FTP es: /pub/kegg/xml/kgml/metabolic/
Una vez alcanzado el directorio metabolic, tenemos a nuestra disposición las carpetas ec (enzimas), ko (ortologías), organisms (organismos) y rn (reaccciones).

De estas cuatro carpetas, la única que no nos interesa es “ko”, ya que no pretendemos el estudio de las diversas familias de especies y por lo tanto su información es irrelevante. En cambio, las enzimas, los organismos y las reacciones son la información que buscamos y que utilizaremos para la extracción de información.

En segundo lugar hay que saber que cada fichero XML situado en cada una de las carpetas, tiene un nombre como “ec00010.xml”, que nos indica que estamos situados en la carpeta de enzimas y que vamos a consultar la estructura del fichero para el pathway 00010. De esta manera siempre sabremos a qué pathway accedemos y qué carpeta es accedida.

Por último, comentar que a nivel de extracción de datos hay cierta información derivada de URLs muy relevante, contenida dentro de estos propios ficheros formando carpetas nuevas que serán parte de nuestra base de datos pero que no están presentes en KEGG.

Capítulo 2
CONSTRUCCIÓN DE LA BASE DE DATOS

Con este capítulo se pretende introducir las tecnologías que se han usado para la construcción de la base de datos y por lo tanto tener en cuenta cuáles son las restricciones que han llevado a usar uno u otro motor de gestión.

MySQL

MySQL es un sistema de gestión de bases de datos que nos proporciona una serie de características muy interesante para ser integrado en el proyecto.

Uno de los pilares básicos es su licencia de software libre que se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia y nos evita cualquier tipo de restricción de copyright para distribución para uso académico. Para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso.

Esta ventaja no tiene ninguna desventaja añadida, ya que la potencia de este motor de gestión viene alabada por la integración en más de 6 millones de equipos y hoy en día es puntero en tecnología de bases de datos relacionales.

La información, por su estructura a nivel biológico, necesita un motor relacional que sea capaz de relacionar diferentes elementos que ocupan un lugar dentro de la jerarquía biológica.

Por ello se ha optado por incluir el motor de almacenamiento InnoDB que nos ofrece varías ventajas que vamos a enumerar para hacernos una idea de la consistencia que supone su adaptación:

- Es una tecnología de almacenamiento de datos de código abierto.
- Soporta bloqueo de registros e integridad referencial.
- Ofrece fiabilidad y consistencia muy superior a antecesores como MyISAM (anterior tecnología de bases de datos)
- Soporta recuperación de problemas volviendo a ejecutar sus logs.
- Combina cachés de registro e índice sin enviar directamente los cambios a las tablas del sistema.

- Almacena físicamente los registros en el orden de la clave primaria, lo que en nuestro caso, supone una enorme ventaja que comentaremos más adelante.

Diseño de la base de datos

Diversos factores han sido tenidos en cuenta a la hora de diseñar la base de datos, entre ellos se encuentra la mayoría de parámetros que vienen dados por la información de KEGG. Por ello, hasta el diseño final, se ha tenido que modificar sucesivamente para satisfacer la necesidad de la información extraída.

Aunque en capítulos posteriores hablaremos de los algoritmos que se han usado para la extracción de datos, es necesario explicar los problemas que ha acarreado KEGG y que en mayor o menor medida, ha podido afectar a la integridad de los datos en la base de datos resultante.

Enumerarlos será de gran ayuda para observar los problemas surgidos y que luego se verán reflejados en la información almacenada:

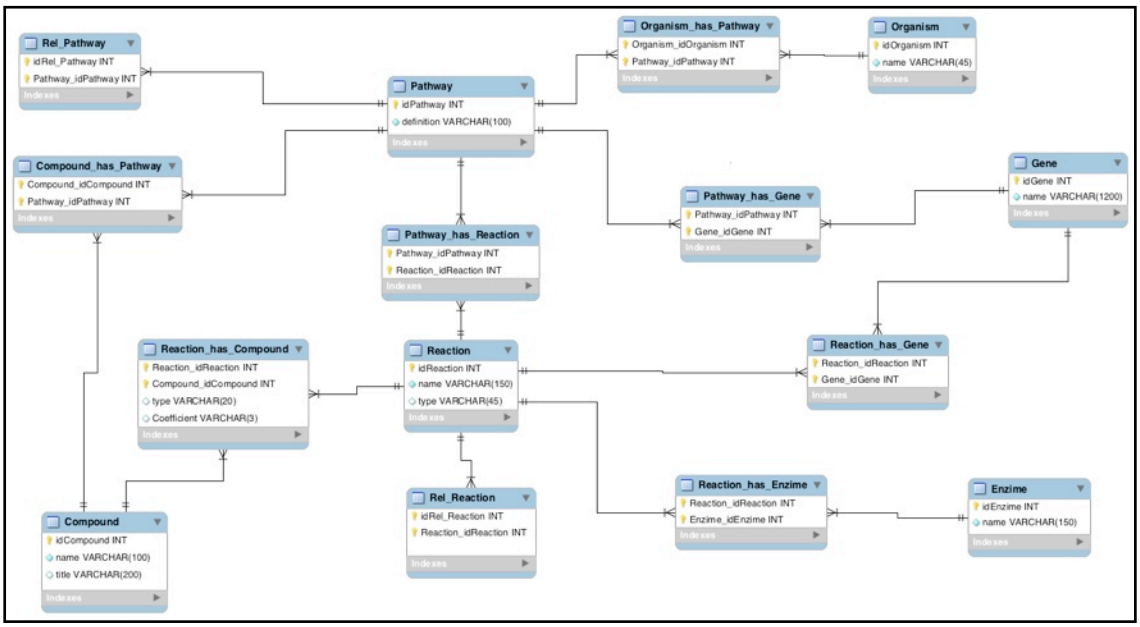
- Inconsistencia: En muchos casos nos encontramos con ciertas enzimas cuyo nombre no es adecuado debido a la falta de precisión de su contenido o de su origen. Un ejemplo claro sería “ec:1.-.-.-” dónde se aprecia claramente que puede pertenecer a la familia de las enzimas que comienzan con numeración 1 pero de la que no se sabe nada más.

- Redundancia: Muchas veces nos encontramos con datos repetidos en varias fuentes de reacciones o compuestos y que teniendo que contener la misma información, difieren. Por ello se ha optado por seleccionar el criterio de aquellos archivos que se muestran al usuario en la página web de KEGG, que a nuestro entender, poseen una mayor fiabilidad al mostrarse al público que accede.

- Incompletitud: Aunque nos encontremos con 5095 compuestos diferentes dentro de la base de datos, sólo 2581 los podemos diferenciar por su nombre químico, por lo que dificulta enormemente identificar estos compuestos por su número de identificación “cpd:C16317”. Este tipo de identificación se encuentra interrelacionada con las otras bases de datos de nucleótidos comentadas en el capítulo 1, pero para poder completar este tipo de información habría que cruzar datos con el consiguiente gasto computacional y temporal, que al fin y al cabo no forma parte del objetivo de este proyecto.

- Longitud de nombres: La mayoría de componentes que se almacenan contiene aproximadamente una media de 100/150 caracteres, que aunque no son pocos, tampoco se puede prescindir de más para su correcta identificación. En cambio, para el caso de los genes se han tenido que hacer una serie de cambios, ya que muchos de los nombres asignados a algunos genes llegaban a ocupar 1150 caracteres, cuando un tipo "varchar" no puede ocupar más de 767 Bytes. Por ello se decidió modificar la columna "name" de la tabla "Gene" a modo indexado para un mejor acceso al contenido sin tener que hacer uso de todo el contenido de la celda, reduciendo considerablemente el tamaño.

En el siguiente gráfico se muestra el diseño final de las tablas y sus columnas correspondientes.



Capítulo 3
HIBERNATE

La base de datos que acabamos de crear debe ser accedida por la aplicación que vamos a desarrollar. Esto supone que todas las tablas devolverán, mediante consultas SQL, la información que hay contenida en ellas.

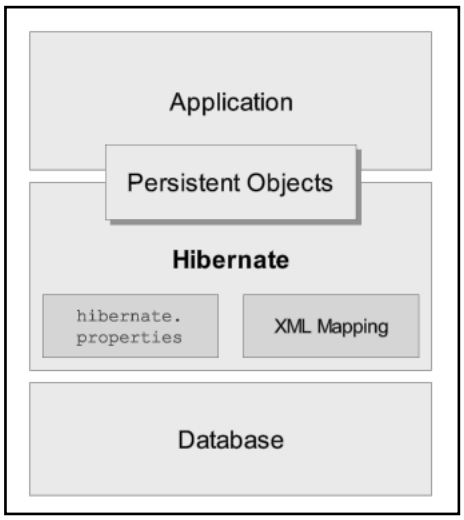
Es muy importante que el método que hemos elegido para su acceso tenga una fuerte interconexión con el lenguaje que vayamos a usar para el desarrollo. Por ello en nuestra opinión no hay mejor método que Hibernate para este tipo de aplicaciones.

Características de Hibernate

Hibernate es una herramienta de Mapeo Objeto Relacional (ORM) para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

Hibernate es software libre, distribuido bajo los términos de la licencia GNU LGPL.

Como todas las herramientas de su tipo, Hibernate busca solucionar el problema de la diferencia entre los dos modelos de datos coexistentes en una aplicación: el usado en la memoria de la computadora (orientación a objetos) y el usado en las bases de datos (modelo relacional). Para lograr esto permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información Hibernate le permite a la aplicación manipular los datos de la base operando sobre objetos, con todas las características de la Programación



Orientada a Objetos (POO). Hibernate convertirá los datos entre los tipos utilizados por Java y los definidos por SQL. Hibernate genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todos los motores de bases de datos con un ligero incremento en el tiempo de ejecución.

Hibernate está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible.
Ofrece también un lenguaje de consulta de datos llamado **HQL** (*Hibernate Query Language*), al mismo tiempo que una "Application Program Interface" (API) para construir las consultas programáticamente (conocida como *"criteria"*).
Hibernate para Java puede ser utilizado en aplicaciones Java independientes o en aplicaciones Java Enterprise Edition (Java EE), mediante el componente **Hibernate Annotations** que implementa el estándar Java Persistence API (JPA) , que es parte de esta plataforma.

Capa "Business"

Cuando nos disponemos a insertar en una base de datos una fila con la información correspondiente en cada columna debemos tener claros que campos debemos rellenar y qué características deben tener cada uno para poder ser admitidas.

Con esta premisa debemos crear las clases que vayamos a utilizar en nuestro programa, que se corresponderán con el número de tablas que tengamos en nuestra base de datos.

Es muy importante que los atributos de cada clase se correspondan con el tipo de datos de la tabla, ya que de lo contrario nos encontraremos con fallos típicos de creación de dichos objetos.
A modo de ejemplo, se explicará una de las tablas que servirá de ejemplo para el resto de capas.

```java
package business;
// Generated 31-oct-2010 20:26:05 by Hibernate Tools 3.3.0.GA

/**
 * Pathway generated by hbm2java
 */
public class Pathway implements java.io.Serializable {

        private int idPathway;
        private String definition;

        public Pathway() {
        }

        public Pathway(int idPathway) {
                this.idPathway = idPathway;
        }

        public Pathway(int idPathway, String definition) {
                this.idPathway = idPathway;
                this.definition = definition;
        }

        public int getIdPathway() {
                return this.idPathway;
        }

        public void setIdPathway(int idPathway) {
                this.idPathway = idPathway;
        }

        public String getDefinition() {
                return this.definition;
        }

        public void setDefinition(String definition) {
                this.definition = definition;
        }

}
```

Observamos como en la definición de clase, perteneciente al paquete “Business”, tenemos los atributos “idPathway” y “definition”. Ambos son columnas en la tabla “Pathway” de la base de datos creada, y por lo tanto sus tipos de datos también son “int” y “string”, debido a que un entero para identificar el identificador del “Pathway” y una cadena de texto para la definición, era lo más conveniente para la especificación.

Los métodos “get” y “set”, son genéricos para darle facilidad a la base de datos a la hora de introducir y recuperar estos atributos de una manera simple y robusta.

En definitiva, cada clase desarrollada tiene un esquema similar adaptándose a las necesidades de los atributos de cada tabla existente.

Mapping XML

En Hibernate, como hemos comentado en el punto de sus características, se nutre de una correspondencia entre los objetos creados a través de Java para poder llevarlos a la base de datos sin necesidad de hacer ningún tipo de "insert" ni de tener en cuenta que motor de almacenamiento estamos usando.

Debido a esta característica, debemos declarar un fichero estructurado XML para cada una de las clases que hayamos definido dentro de la capa vista "Business". Este tipo de ficheros tiene diversas peculiaridades que vamos a ver de manera detallada y cuyo buen funcionamiento con la base de datos depende por completo .

Hay que tener en cuenta que el "mapping" tratará las clases como objetos y nosotros en todo momento vamos a verlos como tal, es decir, nuestra preocupación debe ser la de introducir la información correcta en la clase y crear el objeto. Una vez llegados a este punto, Hibernate a través de los "mapping", se encargará de interpretar este objeto e insertarlo correctamente en la tabla correspondiente.

Para el programador, todas estas acciones ocurren de manera transparente en cada una de las transacciones con la base de datos.

Como en la capa de "Business", usaremos el ejemplo de "Pathway" para ilustrar cada uno de los puntos a destacar.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 31-oct-2010 20:24:55 by Hibernate Tools 3.3.0.GA -->
<hibernate-mapping>
  <class catalog="kegg" name="business.Pathway" table="Pathway">
    <id name="idPathway" type="int">
      <column name="idPathway"/>
      <generator class="assigned"/>
    </id>
    <property name="definition" type="string">
      <column length="100" name="definition"/>
    </property>
  </class>
</hibernate-mapping>
```

Al observar la captura, vemos claramente la estructura ordenada que XML sigue para especificar cada uno de los objetos.

La clave "<hibernate-mapping>" se encargará de indicar que vamos a especificar la estructura que Hibernate interpretará de la información descrita a continuación.

En la clave “<class>” indicaremos la información de la tabla que vamos a especificar y a qué objeto de la capa “Business” corresponde. Es por ello que dentro de “catalog” indicamos el nombre del esquema de la base de datos a la que vamos a acceder, “name” indica el paquete dónde se encuentra el objeto “Pathway” (teniendo clara la sintaxis del punto para indicar “paquete.clase”) y la tabla a la que vamos a acceder dentro de la base de datos.

Cuando ya hemos especificado la clase, comenzamos con la parte más importante dentro del documento, la clave “<id>”. Nos estamos refiriendo a la clave primaria, que nos facilitará la búsqueda de cada uno de los objetos asociado a las diversas columnas y cuya rapidez en la búsqueda se verá influenciado por el tipo de datos que le asignemos. En nuestro caso, la asignación de un entero en la clase “Business” para “idPathway”, fue la manera más sencilla de crear esta clave. Por ello indicamos que “name” en la clase es “idPathway”, “type” en la clase es “int” y todo ello pertenece en la base de datos a la “<column>” con “name” igual a “idPathway”. Es muy importante que dentro de la clave “<generator>” se ponga de manera clara si la clave primaria será asignada de manera manual “assigned” o de manera automática “autoincrement”. Con esto especificamos que las claves, dependiendo del tipo de datos, tengan una correlación o se deje a nuestro criterio por motivos funcionales o prácticos.

Finalmente en la clave “<property>” especificaremos cada una de las “<column>” que no son clave primaria. En este caso al ser una cadena “string”, además del tipo también hay que incluir cual es la longitud que le asignamos dentro de la base de datos, por lo que debería coincidir con el diseño inicial que se hizo para esa tabla.

Capa “Direct Acess Object” (DAO)

A pesar de que es muy difícil poder crear una jerarquía de importancia entre las capas, si que podríamos destacar la capa DAO que abarca este apartado.

La importancia viene dada por el rol de nexo entre la capa “Business” y la capa “Mapping”.

Sin este nexo sería imposible recuperar los objetos de la base de datos o almacenarlos en su caso, con lo que para ello deberíamos explicar de manera detallada cada paso que se sigue desde que establecemos la conexión, hasta que conseguimos la inserción o recuperación.

Comencemos con la conexión del ejemplo que venimos usando dentro de este capítulo, la clase “Pathway”.

```
private final SessionFactory sessionFactory = this.getSessionFactory();

protected SessionFactory getSessionFactory() {
        try {
                SessionFactory session = new Configuration().configure().buildSessionFactory();
                return session;
        } catch (Exception e) {
                e.printStackTrace();
                System.out.println("Session didn´t found");
                throw new IllegalStateException("Could not locate SessionFactory in JNDI");
        }
}
```

El establecimiento de conexión dentro de Hibernate se realiza dentro de un tipo conocido "Session Factory" o Factoría de Sesiones, que no es más que la apertura de una conexión a la base de datos que hayamos configurado previamente a través del fichero de configuración de hibernate que veremos más adelante.

Cada vez que vayamos a realizar algún tipo de acción dentro de la base de datos, una "Session Factory" será creada para gestionar dicha acción. En caso de haber cualquier tipo de error, una excepción será lanzada para advertirnos.

Continuando con el esquema de la capa DAO, varios de los accesos más comunes que se hacen en la base de datos son los de borrado, actualización o inserción. Para no extendernos, explicaremos el funcionamiento de una inserción.

```
public void save(Pathway instance) {
        try {
                Session session = sessionFactory.openSession();
                Transaction tx = session.beginTransaction();
                session.saveOrUpdate(instance);
                tx.commit();
                session.close();
        } catch (RuntimeException re) {
                throw re;
        }
}
```

Como vemos, la "Session Factory" es invocada para realizar la acción de almacenamiento.

La declaración de un tipo "Transaction" para indicar que la "Session Factory" va a comenzar dicha transacción, es la manera de acceder a la base de datos y llevar a cabo acciones con el objeto; claro está, que dicho objeto debe ser un parámetro de la función y por supuesto del tipo "Pathway" en este caso concreto o del tipo que vaya a ser almacenado.

En este punto, usamos "saveOrUpdate" para indicar que queremos almacenar dicho objeto en caso de que exista y en caso contrario que simplemente lo actualice. Vemos de manera clara como en ningún momento debemos de preocuparnos de usar el lenguaje SQL para realizar la inserción y por supuesto llevar a cabo el control de la transacción.

Para ejemplificar hemos usado “saveOrUpdate”, pero cambiando este método por “update” ,”delete” o “save”, obtendríamos de manera secuencial una actualización del objeto o un borrado o un almacenamiento simple.

Tras usar el método deseado para la acción que estemos llevando a cabo, hay que indicarle a Hibernate que haga efectivos los cambios dentro de la base de datos para que se puedan ver reflejados en las tablas, ya que mientras realizamos las transacciones y éstas no se confirman, se quedan en la caché esperando a ser confirmadas. Es por ello que usamos el método “commit” para llevar la confirmación a cabo.

Una vez finalizadas todas las transacciones dentro de la función, simplemente debemos cerrar la conexión para no crear conflictos dentro de los controles de apertura y cierre de conexiones cuando hablamos de cientos de transacciones por segundo.

Por último, hemos dejado la parte más compleja de la capa DAO. Como hemos visto las acciones básicas son bastante triviales en cuanto se conocen los métodos a usar. Pero lo que no es tan trivial son las consultas a la base de datos para recuperar información concreta de una tabla concreta.

Para conseguir este fin tenemos dos caminos para conseguirlo: uno que sigue el esquema de sencillez de Hibernate sin preocuparnos de usar el lenguaje SQL (aunque limitado) u otro que nos obliga a usar un lenguaje diseñado para Hibernate parecido a SQL (denominado HQL) que, aunque es un poco más complejo, le da mucha más potencia a la hora de conseguir resultados más específicos.

Ambos se usan indistintamente según el tipo de consulta que queremos realizar, pero para ir ilustrando con ejemplos empezaremos con el más sencillo.

```
public boolean findById(Integer id) {
        try {
                Session session=sessionFactory.openSession();
                Pathway instance = (Pathway) session.get("business.Pathway", id);
                session.close();
                if(instance == null)
                    return false;
                else return true;
        } catch (RuntimeException re) {
                throw re;
        }
}
```

Este ejemplo es el más transparente a nivel de programación que Hibernate nos ofrece, ya que como hemos dicho anteriormente no debemos de preocuparnos en realizar la consulta en lenguaje SQL. Si dejamos de lado las sentencias que ocupa el tratamiento de la "Session Factory", observamos que al invocar el método "get", simplemente indicando el tipo de objeto que esperamos recibir y su clave primaria, obtenemos el objeto deseado. Evidentemente si queremos realizar cualquier tipo de modificación en ese objeto, es decir, modificar sus atributos, debemos crear un tipo del objeto deseado e igualarlo a la sentencia que se encuentre "get".

El otro camino, más complejo a primera vista, realmente no hace un uso muy diferente del que haríamos si estuviéramos haciendo consultas en SQL.

Para ello Hibernate utiliza un lenguaje de consulta potente (HQL) que se parece a SQL. Sin embargo, comparado con SQL, HQL es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación.

```
public void fillEmptyField(String name,String title){
        try{
                Session session=sessionFactory.openSession();
                Transaction tx=session.beginTransaction();
                Query query = session.createQuery("Select cp.idCompound from Compound cp where cp.name like :name");
                query.setString("name", name);
                int ID = Integer.parseInt(query.list().get(0).toString());
                Compound CPD = (Compound)session.get("business.Compound",ID);
                CPD.setTitle(title);
                session.save(CPD);
                tx.commit();
                session.close();
                //}
        }catch (RuntimeException re) {
                throw re;
        }
}
```

Para ilustrar mejor la opción de consulta por HQL, se ha optado por seleccionar un ejemplo de la clase "Compound" que hace un mayor uso de este mecanismo.

Observamos que la clase "Query" es lo que difiere del método anterior, ya que para poder hacer uso del lenguaje HQL se tiene que invocar al método "createQuery" y almacenar los resultados devueltos.

Una vez tenemos los resultados en nuestro poder, podemos hacer lo que queramos con los objetos. De hecho, en este caso particular, estamos comprobando si uno de los atributos del objeto que estamos buscando en la base de datos se encuentra a "null", es decir, vacío. En caso afirmativo, el "title" pasado por parámetro rellenará este "null" modificando el objeto y devolviéndolo a la base de datos actualizado.

Configuración de Hibernate

Como hemos comentado en todo este capítulo, para el buen funcionamiento de Hibernate debemos especificar cuál es el comportamiento que debe tener basándonos en el motor de la base de datos que usemos y de los "mappings" que hagamos de las clases.

Por ello, se explicará en detalle el fichero entero de configuración para concluir el capítulo dando una especificación completa .

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
                "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
                "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory name="mysessionFactory">
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://127.0.0.1:3306/kegg?autoReconnect=true</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
        <mapping resource="xml/Pathway.hbm.xml"/>
        <mapping resource="xml/Enzime.hbm.xml"/>
        <mapping resource="xml/Compound.hbm.xml"/>
        <mapping resource="xml/Reaction.hbm.xml"/>
        <mapping resource="xml/CompoundHasPathway.hbm.xml"/>
        <mapping resource="xml/PathwayHasReaction.hbm.xml"/>
        <mapping resource="xml/ReactionHasEnzime.hbm.xml"/>
        <mapping resource="xml/ReactionHasCompound.hbm.xml"/>
        <mapping resource="xml/RelPathway.hbm.xml"/>
        <mapping resource="xml/Organism.hbm.xml"/>
        <mapping resource="xml/RelReaction.hbm.xml"/>
        <mapping resource="xml/Gene.hbm.xml"/>
        <mapping resource="xml/OrganismHasPathway.hbm.xml"/>
        <mapping resource="xml/PathwayHasGene.hbm.xml"/>
        <mapping resource="xml/ReactionHasGene.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

La estructura XML es la usada como se puede observar en la figura. El motivo de que sea así, es que, al fin y al cabo,la configuración se trata como si de un "mapping" se tratara, con la diferencia de que debemos especificar ciertos parámetros más.

Los "mapping" que contienen el atributo "resource" son simplemente las rutas en las que se encuentran los objetos correspondientes a las tablas de la base de datos. Por ello en nuestro caso se indica la ruta del paquete "xml" de nuestro programa y después el fichero en el que se encuentra cada uno de ellos. Es importante incluirlos todos pues en caso de no ser así Hibernate pensará que esa tabla no existe y por lo tanto lanzará excepciones que terminarán la ejecución del programa.

Para tratar cada “property” debe ser explicada detalladamente porque la conexión depende de que los datos introducidos estén correctos:

- Driver: El driver será el puente que unirá Hibernate con la base de datos. En este caso MySQL es nuestra base de datos y por lo tanto el driver debe ser compatible con este gestor. Para poder hacer uso de él, en primer lugar, debe estar estar incluido como paquete dentro de nuestro código fuente y posteriormente debe ser indicado en este apartado. Hay que tener en cuenta que este paquete, en el momento de hacer la compilación del programa, debe respetar el sistema operativo en el que estemos trabajando y por ello hay que añadir el paquete para Windows, Linux o MacOsX.

- URL: La url no es más que la dirección local o remota donde se está ejecutando este servidor y por supuesto el nombre de la base de datos que estamos conectando de las que tenga en su lista. En nuestro caso el servidor MySQL se encuentra en local (127.0.0.1), el puerto el estándar de MySQL (3306) y el nombre de la base de datos es “kegg”. Indicar también que la opción “autoReconnect” está activa para modificar el comportamiento de conexión en caso de una caída de ésta para que se reconecte inmediatamente.

- Username: Para poder acceder a una base de datos hay que ser propietario de ciertos permisos que nos darán unos u otros privilegios que dependen única y exclusivamente del administrador del servidor. En nuestro caso el “username” será root y la opción de “pass” se omite ya que no hay contraseña establecida.

- Dialect: El dialecto que “hablemos” con el servidor será el que Hibernate detecte cuando lea el archivo de configuración. Por ello es importante que el motor de almacenamiento que hayamos elegido para nuestro servidor se corresponda con el indicado en Hibernate. La especificación que se ha usado ha sido “MySQLInnoDBDialect” que se corresponde con el motor “InnoDB” que se explicó en el capítulo 2 sobre la construcción de la base de datos.

Aunque hay muchas más propiedades que se podrían haber especificado, las explicadas son las básicas para cualquier base de datos y por lo tanto las usadas para este proyecto.

Capítulo 4
DESCARGA Y EXTRACCIÓN

Este es el capítulo que se encargará básicamente de mostrar los algoritmos de resolución que se han usado para realizar este proyecto y las clases de Java que contienen ciertas instrucciones para el tratamiento y creación de los mismos.

FTP

En el primer capítulo se comentaron las diversas maneras de las que disponíamos para conectarnos a KEGG y extraer la información que necesitábamos para crear la base de datos y concluimos que la mejor manera era a través de su FTP público.

Debido a esta decisión es imprescindible crear un cliente que gestione los ficheros que se deseen descargar y que pueda lanzar comandos para tal fin. Por ello se ha optado por incluir en el proyecto el paquete de conexión de Apache que contiene un cliente FTP con todas las características que buscamos y además continúa con la filosofía de ser software libre y no depender de licencias.

Sus características son las siguientes:

- 100% puro Java, gratis, servidor FTP de código abierto.
- Soporte multiplataforma y diseño multihilo.
- Directorio virtual de usuario, permisos de escritura, tiempo de inactividad y soporte de limitación de ancho de banda de subida/bajada.
- Soporte para usuario anónimo.
- Soporte para remotar ficheros de subida y descarga.
- Maneja transferencias de datos en ASCII y binario.
- Soporta restricción de IPs para baneo.
- Bases de datos y archivos pueden ser usados para almacenar datos de usuario.
- Todos los mensajes FTP pueden ser customizados.
- Soporte SSL/TLS implícito y explícito.
- Soporte MDTM - los usuarios pueden cambiar la fecha incluida en los ficheros.
- Soporta "MODE Z" para mayor velocidad de transferencia en subida/bajada.
- Administrador customizable, restricción de IP, monitor de actividad añadido fácilmente .
- Notificación de eventos de usuario (Ftplet).

Con las características mencionadas, el uso del FTP se simplifica mucho a la hora de establecer las conexiones con KEGG. Para no extendernos demasiado, se comentarán los elementos más importantes que serán usados posteriormente y el esquema general del algoritmo principal.

La idea principal que se pretende es descargar los XML correspondientes a las reacciones, enzimas y organismos que hay en el FTP de KEGG. Tras realizarse la descarga, se "parsearán" los ficheros y se extraerá la información para su almacenamiento. Para comenzar con el primer paso debemos establecer la conexión remota con el servidor. Por ello se mostrará una figura de la conexión y posteriormente se mostrarán los comandos usados para navegar y descargar la información.

```
      //Socket created
FTPClient f_client = new FTPClient();

public void connect() {
    try {
        int reply;
        f_client.connect("ftp.genome.jp");
        f_client.login("anonymous", "");
        // After connection attempt, you should check the reply code to verify success.
        reply = f_client.getReplyCode();

        if (!FTPReply.isPositiveCompletion(reply)) {
            f_client.disconnect();
        }
    } catch (IOException e) {
        if (f_client.isConnected()) {
            try {
                f_client.disconnect();
            } catch (IOException f) {
                // do nothing
            }
        }
        System.err.println("Could not connect to server.");
    }
}
```

El tipo "FTPClient" proporcionado por el paquete Apache nos permite de una manera muy sencilla crear un socket cliente para la conexión. Con los métodos "connect" y "login", conseguimos establecer la conexión y con "getReplyCode" nos aseguramos que todo ha sido correcto y podemos empezar a mandar órdenes al servidor FTP. En caso de ocurrir cualquier error, observamos como el manejo se basa en la desconexión mediata mediante el método "disconnect" y la posterior notificación al usuario.

Incluir un ejemplo de los algoritmos usados para cada uno de los elementos mencionados anteriormente no sería muy ilustrativo por la cantidad de código que requieren. Por lo tanto se mostrará solo parte del código y en este apartado simplemente se mencionarán las acciones que se realizan para cada uno de ellos y las mejoras implementadas para una mayor eficiencia del tiempo de usuario en la descarga de los ficheros.

Para comenzar la descripción se mostrará un pseudocódigo general y posteriormente sus características.

```
Cambiar directorio en FTP → changeWorkingDirectory
Recuperar nombres de archivos en FTP → listNames
              si carpeta existe → no hacer nada
                ➟  sino crear carpeta
Recuperar nombres de archivos en directorio local
Borrar archivos de local que ya no se encuentran en FTP  → Delete_Ghost
Recuperar nombres de archivos en directorio local actualizado
Borrar archivos incompletos o vacios → Recovery
Descargar archivos resultantes...
```

Cambiar el directorio FTP es la instrucción básica para poder moverse por el árbol de directorios de KEGG y llegar a la carpeta que nos interesa.

Recuperar los nombres de los archivos que se encontraban en el FTP es imprescindible si queremos copiar la información de dicho fichero con el mismo nombre que le ha dado KEGG en su base de datos. Pero aquí se crea una problemática que tuvo que ser resuelta según se le iban añadiendo mejoras al algoritmo. En caso de que hubiéramos interrumpido la descarga por cualquier motivo, sería muy costoso empezar a descargar todo desde el principio. Por ello convenía gastar algo de tiempo haciendo unas cuantas comprobaciones y finalmente descargar los archivos faltantes.

Antes de la descarga, se debe crear la carpeta cuyo contenido se dispone a descargar. Por ello comprobar que ya está creada nos permite saber que por lo menos un intento de descarga ha habido previamente.

Recuperar los nombres de los archivos de nuestro directorio local nos permite cruzarlos con los nombres del FTP y así averiguar si entre la última descarga y ahora nos encontramos con el problema de que mantenemos información desactualizada en nuestra base de datos que debe ser eliminada. Para este fin se creó el método “Delete_Ghost”, que significa “Borrar fantasma”.

Pero el verdadero problema viene cuando además de poder tener archivos desactualizados que también estén corruptos o que también estén vacíos debido a una interrupción de la conexión. Solucionar este problema se consigue inspeccionando el contenido y marcando unos mínimos

de tamaño por archivo que nos aseguran su integridad. El método “Recovery”, que significa “Recuperación”, se encargará de devolver una lista limpia de nombres de ficheros preparada para ser descargada del FTP.

Hay que apuntar que la base de datos de KEGG se encuentra en Japón y que por lo tanto el tiempo de conexión y descarga tarda el equivalente a un archivo, multiplicado por todos aquellos que nos resten para completar la descarga; que en caso de ser desde cero, puede llegar a tardar entre 4 o 5 días ininterrumpidos con un tamaño de descarga de 6 GigaBytes aproximadamente. Por ello es importante poder interrumpir la descarga y reanudarla.

HTML

Dentro de la información contenida en KEGG, hay ciertos atributos de clases que no podemos extraer de los XML descargados previamente. Por ello debemos acceder a información complementaria descrita en lugares diferentes y por lo tanto no tan accesible.

Los casos de los que estamos hablando son: la extracción de las ecuaciones estequiométricas y la extracción de los nombres químicos de componentes.

En ambos casos, la información que nos proporcionaban los XML de las reacciones y los compuestos no conseguía completar los requisitos que se solicitaban para la creación de la base de datos. Fue entonces cuando se recurrió a los enlaces URL de dichos ficheros que nos redireccionaban a páginas HTML que contenían la información que buscábamos.

El proceso en parte es bastante sencillo: Tanto en los XML de reacciones como en los de compuestos, estos enlaces URL se identificaban por el nombre de cada elemento descrito; con lo que tras una extracción previa los enlaces, asociándolos a identificadores, podríamos descargar la página HTML completa y posteriormente extraer la información requerida.

Hay que especificar que los HTML no son más que XML con ciertos cambios en su estructura descriptiva. Por lo tanto la extracción de datos no diferirá excesivamente de la extracción de ficheros XML normales.

En la figura podemos observar la descarga de los HTML de reacciones. La creación de un tipo “url” invoca al método “HTML.get”, accede al atributo “url_chain” y realiza la apertura de conexión y el posterior volcado del fichero a través del método “pipe”.

Apuntar que las mejoras descritas en el apartado anterior son extensibles para la descarga de los HTML.

```
//URL Connection opened
try {
    if(!Compare(F_names,HTML.get(i).name+".html","equations")){
    url = new URL(HTML.get(i).url_chain);
    urlCon = url.openConnection();
    //File download
    //Access to web content
    urlCon.setDoOutput(true);
    InputStream in = null;
    in = url.openStream();
    pipe(HTML.get(i).name,in,"equations");
    }
} catch (Exception e) {
    System.out.println("Downloading HTML error");
}
```

Extracción de datos o "Parsing"

Hasta ahora hemos podido ver cómo uno de los primeros pasos para comenzar a almacenar información en la base de datos era realizar la descarga de los archivos que la contenían. Pero la parte más importante comienza justo después de la descarga de estos ficheros ya que, aunque la información esté contenida en ellos, se tiene que extraer de una manera ordenada y precisa para poder manipularla más adelante y que ésta sea válida.

Es ahora cuando entra dentro de este contexto el proceso de "parsing" o extracción. Se ha comentado que los ficheros descargados tienen una estructura específica que nos facilita la búsqueda e identificación de la información. Pero para poder analizar correctamente la estructura es preciso acudir a herramientas que poseen métodos en forma de librería para trabajar con java y por lo tanto adelantar bastante trabajo.

Estamos hablando de la API para el procesamiento de XML (JAXP) que permite a las aplicaciones extraer, transformar, validar y consultar documentos XML usando una API que es independiente de la implementación del procesador XML que se use. JAXP proporciona una capa de conectividad que permite a los desarrolladores crear sus propias implementaciones sin introducir dependencias en el código de la aplicación. Usando este software, desarrolladores de aplicaciones y herramientas pueden construir aplicaciones en Java completamente funcionales a nivel XML para el propósito que se crea conveniente.

Por otra parte, también se ha usado una API específica para la extracción de información de los HTML. Esta en particular se denomina "Jericho HTML Parser" y es una biblioteca Java que permite el análisis y la manipulación de partes de un documento HTML, incluyendo etiquetas de servidor, al reproducir textualmente el código HTML no reconocido o no válido. También proporciona métodos de alto nivel para manipular código HTML.

Apuntar que se trata de una biblioteca de código abierto distribuida bajo la licencia LGPL.

Se van a exponer un par de ejemplos de cada una de las bibliotecas donde se podrá visualizar el proceso del tratamiento de la información en el momento de extraerse y almacenarse a través de Hibernate.

JAXP

Comenzaremos hablando de los archivos XML descargados previamente y el uso de esta librería con un ejemplo ilustrativo.

Como en apartados anteriores, se van a mostrar pseudocódigos con el fin de clarificar el funcionamiento, aunque posteriormente se podrá consultar el código fuente completo en el apéndice.

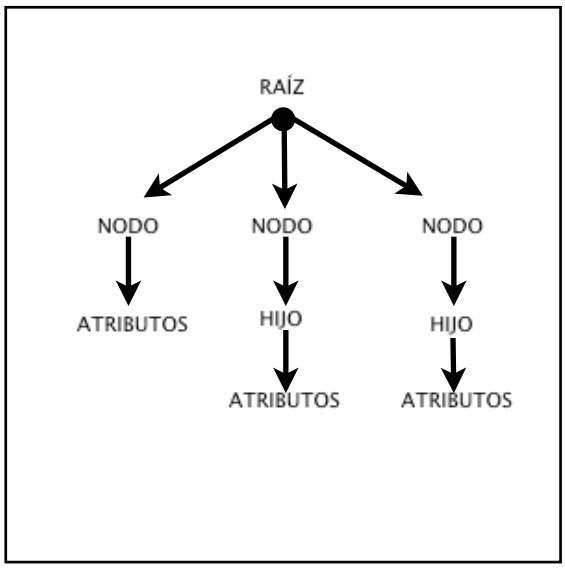
Comprender la estructura de los "parsers" no es tarea compleja con ciertos conocimientos sobre estructuras de árboles. Pero para que nos hagamos una idea del funcionamiento principal hay que imaginar el documento XML como un árbol cuya raíz es la clase principal y el resto, al fin y al cabo, con hojas o hijos del mismo. Por lo tanto, siguiendo este esquema se expondrá un ejemplo de lo que nos encontramos en un comienzo y como lo tratamos posteriormente.

```
<?xml version="1.0"?>
<!DOCTYPE pathway SYSTEM "http://www.genome.jp/kegg/xml/KGML_v0.7.1_.dtd">
<!-- Creation date: Sep 30, 2010 09:16:33 +0900 (GMT+09:00) -->
<pathway name="path:ec00010" org="ec" number="00010"
         title="Glycolysis / Gluconeogenesis"
         image="http://www.genome.jp/kegg/pathway/ec/ec00010.png"
         link="http://www.genome.jp/kegg-bin/show_pathway?ec00010">
    <entry id="13" name="ec:4.1.2.13" type="enzyme" reaction="rn:R01070"
        link="http://www.kegg.jp/dbget-bin/www_bget?4.1.2.13">
        <graphics name="4.1.2.13" fgcolor="#000000" bgcolor="#BFBFFF"
            type="rectangle" x="483" y="404" width="46" height="17"/>
    </entry>
    <entry id="37" name="ec:1.2.1.3" type="enzyme" reaction="rn:R00710"
        link="http://www.kegg.jp/dbget-bin/www_bget?1.2.1.3">
        <graphics name="1.2.1.3" fgcolor="#000000" bgcolor="#BFBFFF"
            type="rectangle" x="289" y="943" width="46" height="17"/>
    </entry>
    <relation entry1="71" entry2="74" type="ECrel">
        <subtype name="compound" value="87"/>
    </relation>
    <relation entry1="71" entry2="73" type="ECrel">
        <subtype name="compound" value="87"/>
    </relation>
    <relation entry1="69" entry2="71" type="ECrel">
        <subtype name="compound" value="87"/>
    </relation>
    <reaction id="79" name="rn:R02187" type="irreversible">
        <substrate id="88" name="cpd:C00221"/>
        <product id="90" name="cpd:C01172"/>
    </reaction>
    <reaction id="80" name="rn:R05132" type="irreversible">
        <substrate id="106" name="cpd:C06186"/>
        <product id="107" name="cpd:C06187"/>
    </reaction>
    <reaction id="81" name="rn:R04394" type="irreversible">
        <substrate id="109" name="cpd:C01451"/>
        <product id="108" name="cpd:C06188"/>
    </reaction>
```

La figura mostrada arriba no pertenece a ningún compuesto real ya que todos superan las 500 líneas de código y por lo tanto es ilegible. En su lugar se ha mostrado un montaje de lo que serían las 3 partes de las que constan los archivos XML.

Retomando la estructura de árbol, XML nos proporciona la etiqueta “pathway” como raíz del árbol y partiendo desde ahí nos encontramos con los hijos “entry”, “relation” y “reaction”. Este patrón se repite en todos los archivos y gracias a ello se puede automatizar la extracción. También observamos como dentro de los hijos nos encontramos hijos a su vez como “graphics” para “entry” y “substrate” junto con “product” para “reaction”. Todos estos “tags” o etiquetas marcan la información que las acompaña y por lo tanto la identifica.

Los atributos se encuentran dentro de estos nodos y deberemos recorrerlos para acceder a sus valores.



Llegados a este punto, ya podemos mostrar el funcionamiento del “ parser” a la hora de construir el árbol de nodos que viene indicado en la figura de arriba. Hay ciertos puntos de los algoritmos de extracción que aunque parezcan obvios no lo son sino se entiende la procedencia de las estructuras que planteamos. Es por ello por lo que se insiste tanto en el entendimiento del mismo para poder implementar futuras mejoras en el mecanismo de acceso y consulta de información en los nodos.

La figura siguiente muestra cómo se accede a lo que se acaba de explicar y por lo tanto el tratamiento de los nodos a través del algoritmo.

```
//Creating object file
File file = new File("./xml/ec/" + list[d].getName());

//New Document factory
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(file);
doc.getDocumentElement().normalize();

//Pathway number & title => TO DB
int path = Integer.parseInt(((Element) doc.getElementsByTagName("pathway").item(0)).getAttribute("number"));
PATH = new Pathway(path, ((Element) doc.getElementsByTagName("pathway").item(0)).getAttribute("title"));
PATH_H.save(PATH);
//Filling HashTable with Ids
ID_extraction(doc);

//Compounds & Enzimes => TO DB
compoundAndenzimes(doc, path);
```

El tipo “DocumentBuilderFactory” es el encargado de construir el árbol descrito gracias al método “parse”, donde podemos observar claramente cómo se le pasa el archivo que pretendemos analizar.

Una vez normalizado, acceder a sus campos es sencillo. Simplemente accediendo a través del método “getElementsByTagName” indicamos la etiqueta de la que queremos extraer atributos y aplicamos “getAttribute” con el nombre del atributo que deseamos. Teniendo el valor o dato en una variable, haremos las acciones deseadas que en nuestro caso es almacenarlo en la base de datos.

La función “compoundAndenzimes” será la encargada de recorrer los nodos y automatizar el proceso, ya que en el párrafo anterior accedíamos simplemente a información de la raíz que se encuentra disponible en el momento de aplicar el “parser”.

```
public void compoundAndenzimes(Document doc, int path) {
    //Creating Compound Persistent Session
    NodeList nodeLst = doc.getElementsByTagName("entry");
    if (nodeLst != null) {
        //All titles in XML
        for (int s = 0; s < nodeLst.getLength(); s++) {
            Node fstNode = nodeLst.item(s);
            if (fstNode.getNodeType() == Node.ELEMENT_NODE) {
                Element T = (Element) fstNode;
                //GETTING COMPOUND ATTRIBUTES
                if (T.getAttribute("type").compareToIgnoreCase("compound") == 0) {
                    //Checking Compound Exists
                    if (!CMP_H.findByField(T.getAttribute("name"))) {
                        CMP = new Compound(T.getAttribute("name"));
                        CMP_H.save(CMP);
                    }
                    //Save Compound-Pathway relation
                    CMP_PATH = new CompoundHasPathway(CMP_H.findByName(T.getAttribute("name")), path);
                    CMP_PATH_H.save(CMP_PATH);
                }
                //GETTING ENZIME ATTRIBUTES
                if (T.getAttribute("type").compareToIgnoreCase("enzyme") == 0) {
                    //Checking Enzime Exists
                    if (!ENZ_H.findByField(T.getAttribute("name"))) {
                        ENZ = new Enzime(T.getAttribute("name"));
                        ENZ_H.save(ENZ);
                    }
```

Esta función hace uso de diversos métodos para recorrer el árbol según nuestras preferencias.

Observando el algoritmo de arriba a abajo, vemos como creando una lista con los elementos del árbol cuya etiqueta se llama “entry”, nos permite recorrer todos los elementos seleccionados y comprobar si nos encontramos en un nodo cuyo atributo es el tipo “compound” o el tipo “enzyme”.

Teniendo clara la metodología de extracción, ahora sólo nos queda ver dentro del mismo código la relación con Hibernate. Observamos que una vez situados en el nodo que queremos y teniendo el atributo localizado, simplemente pasándole este atributo por parámetro a la función de la variable DAO "findByField", conseguimos averiguar si ya está insertado este elemento en la base de datos o ,por el contrario ,debemos crear uno nuevo y posteriormente almacenarlo mediante el método "save". Como vimos en el capítulo de Hibernate, todos estos métodos tuvieron que ser descritos en las capas DAO previamente.

Jericho HTML Parser

Hemos visto como JAXP trata los datos en forma de nodos para poder recorrer la información de un documento XML, pero para poder tratar documentos HTML necesitamos un "parser" específico que nos permita realizar operaciones más concretas de las que nos proporciona JAXP. Por ello es por lo que mostraremos un pequeño ejemplo de sus funciones sin profundizar mucho, ya que como hemos comentado anteriormente, la estructura de un documento HTML es al fin y al cabo, la misma que la de cualquier documento XML.

```
if (n_file.endsWith(".html")) {
    //Creating object file
    String file = "./xml/equations/" + n_file;
    System.out.println(file);
    FileInputStream IN = new FileInputStream(new File(file));
    Source source = new Source(IN);
    List<net.htmlparser.jericho.Element> Tr = source.getAllElements(HTMLElementName.TR);
    //Clean reaction html extension
     String reaction = Arrays.asList(n_file.split(".html")).get(0).toString();
     String[] reactions = reaction.split(" ");
     List R = Arrays.asList(reactions);
     //Index Reactions
     int r=0;
    for (net.htmlparser.jericho.Element linkTR : Tr) {
        String EQ = linkTR.getTextExtractor().toString();
        if (EQ.contains("Equation") && !EQ.contains("Reaction")) {
            String[] Parts = EQ.split(" ");
            equation_DB(R.get(r).toString(), Parts);
            r++;
        }
    }
}
```

Estas funciones específicas de las que se hablaba son "getAllElements" y "getTextExtractor" que como podemos ver en el código situado arriba, ambas pertenecen a la clase "net.htmlparser.jericho.Element".

Con la primera función conseguimos almacenar en una lista todos los nodos cuya etiqueta comience por "TR", que es una etiqueta especial de HTML. Tras manipular información de otros campos, simplemente convertimos cada uno de los elementos de la lista en texto plano y después de procesarlos con diversas funciones, almacenamos el contenido como se ha visto anteriormente con JAXP.

Capítulo 5
ESTRUCTURA DE ALGORITMOS

El presente capítulo pretende mostrar la ingeniería del software que hay detrás de la aplicación del proyecto. Hasta ahora podemos resumir que todos los conceptos tratados formaban parte de la base de la aplicación y de conexión de la base de datos.

Ahora nos vamos a centrar en los algoritmos y estructuras que hemos usado para llegar a los resultados y conclusiones que este proyecto pretendía. Hay que tener claro que aunque todos los algoritmos que se presenten resuelven problemas de menor o mayos envergadura, el que realmente supondrá las conclusiones finales con fines de investigación, será el algoritmo de búsqueda de reacciones en cadena que se explicará en este capítulo.

Tablas Hash y partición de cadenas

Dos de las estructuras más recurrentes en este proyecto son las tablas hash y la partición de cadenas de texto o "strings".

Una tabla hash o mapa hash es una estructura de datos que asocia llaves o claves con valores. La operación principal que soporta de manera eficiente es la búsqueda: permite el acceso a los elementos almacenados a partir de una clave generada (en nuestro caso un valor dado). Funciona transformando la clave con una función hash en un hash, un número que la tabla hash utiliza para localizar el valor deseado.

Las tablas hash se suelen implementar sobre vectores de una dimensión, aunque se pueden hacer implementaciones multi-dimensionales basadas en varias claves. Como en el caso de los arrays, las tablas hash proveen tiempo constante de búsqueda promedio O(1) sin importar el número de elementos en la tabla. Sin embargo, en casos particularmente malos el tiempo de búsqueda puede llegar a O(n), es decir, en función del número de elementos. Comparada con otras estructuras de arrays asociadas, las tablas hash son más útiles cuando se almacenan grandes cantidades de información.

```
Hashtable ListSP = new Hashtable();
ListSP.put(i, item);
SubsProd SP = (SubsProd) ListSP.get(lsm.getMaxSelectionIndex());
```

En este ejemplo, teniendo en cuenta que el editor lo marca en rojo por ser sintácticamente incorrecto, observamos la creación de una tabla hash en Java, la inserción de una estructura "item" y una clave "i". Posteriormente recuperamos la estructura "item" ingresando la clave

correspondiente y forzando el tipo a la estructura correspondiente ya que la tabla hash siempre nos devuelve un tipo genérico “object”.

La partición de cadenas de texto debida a que la mayoría de información que debemos manipular contiene texto y no siempre viene de la manera que deseamos almacenarla.

Por ello más que mostrar un tipo de estructura de datos, se va a mostrar el algoritmo que se ha usado para la partición de cadenas a la hora de reconstruir las ecuaciones estequiométricas de las diversas reacciones que se muestran en la aplicación.

```
for (int j = 0; j < Subs.size(); j++) {
    String s1 = CPD_H.Title(Integer.parseInt(Subs.get(j).toString()));
    if (j + 1 == Subs.size()) {
        screenList = screenList + " " + s1 + " <=> ";
    } else {
        screenList = screenList + " " + s1 + " + ";
    }
}
for (int j = 0; j < Prod.size(); j++) {
    String s2 = CPD_H.Title(Integer.parseInt(Prod.get(j).toString()));
    if (j + 1 == Prod.size()) {
        screenList = screenList + " " + s2;
    } else {
        screenList = screenList + " " + s2 + " + ";
    }
}
JL.add(screenList);
```

Observamos como la función va a acumular los nombres de los diversos compuestos en sustratos y productos para mostrarlos finalmente en la cadena “screenList”. El algoritmo se basa en reconocer en qué punto de la cadena nos encontramos para saber que almacenar en cada momento y si además debemos incluir algún símbolo entre compuesto y compuesto.

Algoritmo de búsqueda de reacciones en cadena

Este punto debe explicar en qué basa el algoritmo su funcionamiento porque sin entenderlo sería imposible explicar la metodología empleada.

La base de este proyecto pretendía dos cosas: La primera gestionar y construir la base de datos, cuestión que ya hemos resuelto en los capítulos anteriores, y la segunda desarrollar un algoritmo que nos permita encontrar las reacciones en cadena necesarias para producir un compuesto específico, que se va a explicar a continuación.

Para comprender bien en qué consiste la búsqueda de estás reacciones en cadena, hay que introducir el concepto de reacción estequiométrica. Una reacción estequiométrica no más que una suma de compuestos que producen otros.

$$2\,H_2 + O_2 \rightarrow 2\,H_2O$$

Los compuestos que se encuentran en la parte izquierda de la reacción se conocen como sustratos y los que se encuentran en la parte derecha se conocen como productos. En ambas partes estos compuestos pueden tener coeficientes que indican la cantidad de moléculas de dicho compuesto y la forma de la flecha indica si la reacción es reversible o irreversible. En el caso mostrado arriba es irreversible, ya que 2 moléculas de agua más una de oxígeno nos proporciona 2 moléculas de agua. Pero en cambio, las dos moléculas de agua no nos pueden proporcionar hidrógeno y oxígeno por separado.

Explicadas las reacciones ahora sólo queda comentar en detalle lo que buscamos. Así que supongamos que estamos interesados en cierto producto que se da en cierta reacción perteneciente a un organismo. Ahora debemos coger este compuesto, que es producto de una reacción, y buscarlo como sustrato en reacciones de otros organismos. Todo ello con el fin de encontrar un compuesto en concreto, que es el compuesto objetivo del problema.

Para visualizarlo de una manera más simple supongamos el siguiente ejemplo:
Conocemos un elemento denominado “Phosphoenolpyruvate” que forma parte del conjunto de productos de una reacción de un organismo “aaa”.
Queremos producir, partiendo de ese elemento, otro denominado “Phosphoacetaldehyde”. Pero con la restricción de que sea en otro organismo diferente de “aaa”.

El resultado del programa sería:

```
2-Phospho-D-glycerate <=> Phosphoenolpyruvate + H2O
Phosphoenolpyruvate <=> 3-Phosphonopyruvate
3-Phosphonopyruvate <=> CO2 + Phosphonoacetaldehyde
```

que como podemos observar nos proporciona una cadena en la que el primer elemento forma parte de la primera reacción como producto y como sustrato de la segunda reacción. La producción de “3-Phosphonopyruvate” nos sirve de enlace para alcanzar el producto deseado en la tercera reacción.

Una vez clara la parte biológica, vamos a abarcar la parte técnica que se encargar de todo este proceso de búsqueda y qué tipo de estructura de datos se ha usado para conseguirlo.

Hay que comenzar diciendo que la estructura usada para la resolución de la búsqueda es la recursión de la pila del sistema. La recursión se usa para la resolución de problemas que tienen cierta estructura que facilita mucho su planteamiento y resolución.

Para el caso que se está tratando hay que indicar que la base de datos nos puede devolver todas las reacciones asociadas a un organismo, pero todas estas reacciones presentan un problema.
El problema consiste en que cada compuesto puede ser producto en una reacción y sustrato en otra por lo que, a no ser que se acote de alguna manera la búsqueda, ésta puede llegar a ser infinita y por lo tanto no encontrar ninguna solución factible.

Para resolverlo se ha optado por solicitarle al usuario a través de la interfaz el número de reacciones en cadena que está dispuesto a asumir con el fin de acotar la recursión. Esto quiere decir que cuando lancemos el algoritmo, éste se encargará de no enlazar más de X reacciones elegidas por el usuario, de manera que contemos con un caso base que nos indica que en caso de no haber encontrado una solución a esa profundidad, entonces deseche el camino recorrido y comience con la siguiente opción.

Se va a exponer el algoritmo recursivo para que ser pueda apreciar mejor lo que se explica en los párrafos anteriores.

```
public List recursion(int steps,int R,List REACTION, int i){
    List RES = new ArrayList();
    if(steps == 0 || i > REACTION.size()-1){
        RES.add(-1);
        return RES;
    }

        if(RCT_CPD_H.reactProds(R).contains(Y)){
            RES.add(R);
            return RES;
        }
        else{
            List Intersection = RCT_CPD_H.reactSubs(Integer.parseInt(REACTION.get(i).toString()));
            Intersection.retainAll(RCT_CPD_H.reactProds(R));

            if(Intersection.size()>0){
                 RES.add(R);
                 RES.addAll(recursion(steps-1,Integer.parseInt(REACTION.get(i).toString()),REACTION,i+1));
                 return RES;
            }
            else{
                return recursion(steps,R,REACTION,i+1);
            }
        }
}
```

La función de recursión tiene como tarea realizar llamadas a sí misma hasta llegar a los casos base y así ir resolviendo cada instancia o llamada realizada. Vemos cómo en la llamada los parámetros usados son: número de reacciones máximas, reacción actual que está siendo revisada, lista de reacciones totales e índice para recorrer la lista de reacciones totales.

Sólo en caso de haber llegado al tope de reacciones máximas o excedernos en el tamaño del índice que usamos para movernos por la lista de reacciones, alcanzamos el caso base, que consiste en agregar el valor -1 y devolver la lista creada en esa llamada.

El resto de la función comprueba si hemos alcanzado el producto deseado y en caso de no ser así actualizar las lista de reacciones totales al caso de estudio actual y modificar la siguiente instancia de la llamada a la función.

Capítulo 6
INTERFAZ GRÁFICA

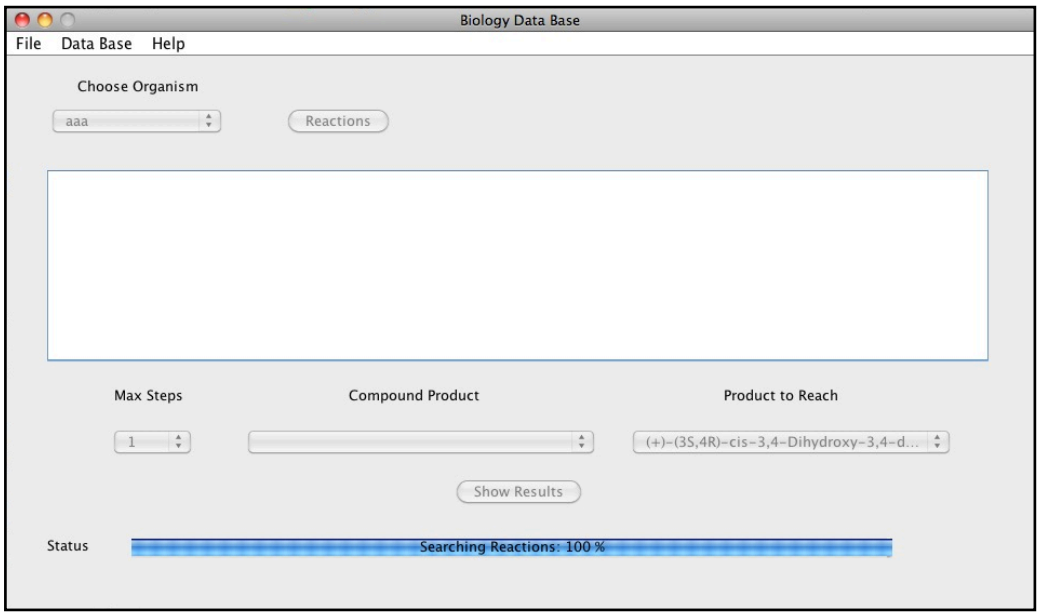
La interfaz gráfica es uno de los elementos cruciales de este proyecto. Parte de su buen funcionamiento depende de cómo el usuario final pueda interactuar con ella. Una de las ventajas indiscutibles de esta parte del proyecto es que todo el funcionamiento interno junto con la parte visual son capaces de ejecutarse en cualquier sistema con Windows, Linux o MacOsX, siempre y cuando se cumplan ciertos requisitos, teniendo en cuenta que la interfaz no se verá afectada en ninguno de los sistemas ni afectará a su efectividad en cuanto a ejecución.

Elementos y diseño

Como comentábamos el usuario cuando maneja una aplicación no tiene porque entender su funcionamiento a simple vista, por ello la interfaz gráfica debe disponer de elementos lo más intuitivos posibles para que al menos se pueda defender en los inicios.

Estos elementos pueden ser tanto informativos como accionadores. Los elementos informativos pretenden que el usuario cuando ejecuta una función pueda observar el avance de la misma sin llevarse la impresión de que el programa está teniendo un mal funcionamiento.

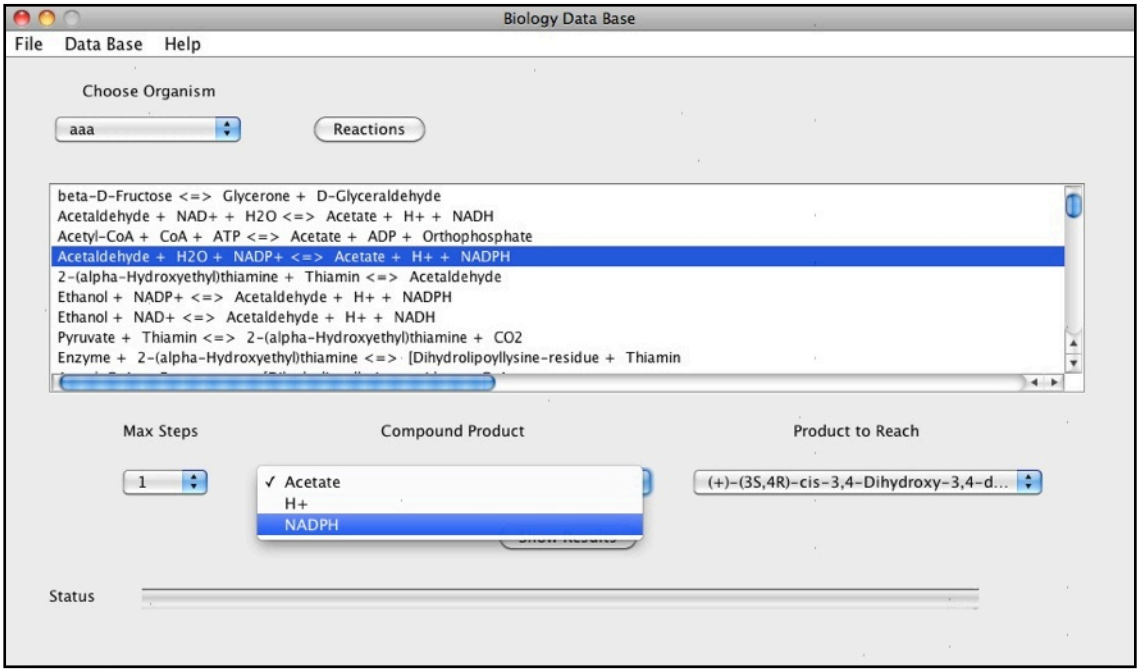
La resolución de este tipo de problema se solventa implementando una barra de progreso en la que se indica el porcentaje de tarea realizada y la acción que se está realizando. Es una solución muy efectiva porque nos informa que la aplicación está haciendo su trabajo y estima aproximadamente el tiempo que tardará en completarlo.



Otra manera de informar al usuario, un tanto menos directa, es haciéndole saber que ciertas acciones no estarán disponibles hasta que no se lleven acabo otras previas. Esto se consigue deshabilitando botones u opciones según se interactue con la interfaz. Este efecto se puede observar en la figura de arriba, cuyos elementos aparecen en gris mientras se ejecuta la opción de búsqueda de reacciones para cierto organismo.

Los elementos accionadores sirven para interactuar directamente con el usuario con el fin de ejecutar partes del código interno y que nos retorne un resultado en base a las opciones indicadas.

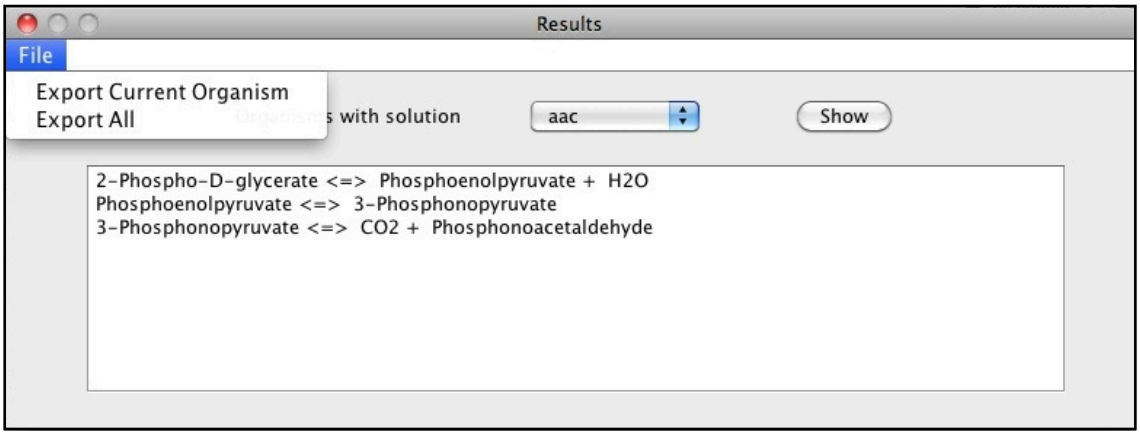
Dentro de la aplicación nos encontramos con botones, listas desplegables, listas de selección y opciones de barra de herramientas. Todas ellas tienen una acción asignada y todas ellas se pueden ejecutar siempre y cuando se den ciertas condiciones, como comentábamos anteriormente.



Observamos las listas despegables disponibles para selección de organismo, profundidad de búsqueda, selección de producto de la opción seleccionada y producto que se desea producir.

Ésta es una manera muy sencilla de darle opciones al usuario sin requerir la comprobación de datos introducidos manualmente, que terminan provocando fallos y ralentizaciones que repercuten en el manejo.

Los desplegables de barras de herramientas tienen un uso parecido y llevan a cabo opciones principales o extra.

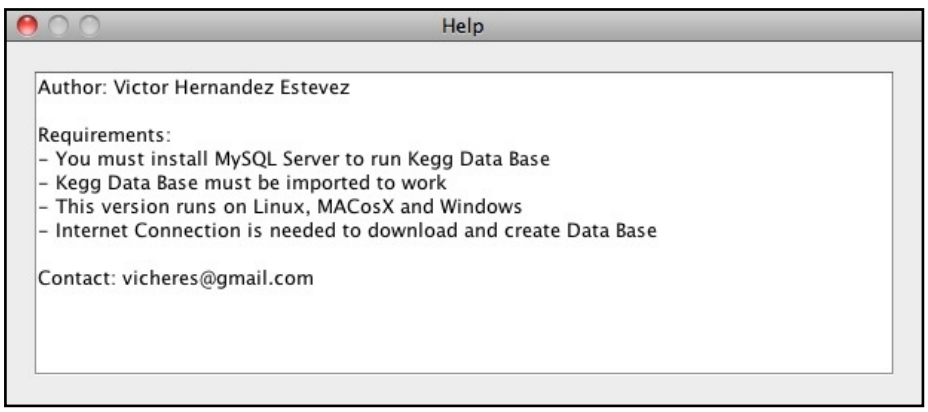


Ventanas como la expuesta arriba nos indican la visualización de la que hablábamos en el párrafo anterior. En este caso se nos da la opción "extra" de exportar los resultados que vemos en pantalla del organismo seleccionado o exportar todos los organismos que nos han proporcionado resultados factibles.

La otra posibilidad es que se nos proporcionen por medio de los desplegados de barras de herramientas opciones principales para descargar la base de datos y posteriormente construirla.

Los otros elementos restantes son los botones, que debido a su uso cotidiano no necesitan presentación. Aunque se puede decir que en nuestro caso ejecutan las tareas importantes como la búsqueda de reacciones de un organismo o la búsqueda de reacciones en cadena.

Hay ciertas ventanas de la interfaz que aunque se den por triviales no está de más revisarlas. Estamos hablando de las ventanas de requisitos y ventanas de acceso al sistema de ficheros del ordenador.



Ambas ventanas son necesarias y por lo tanto forman parte de la interfaz. La ventana de requisitos pretende exponer las condiciones para que el funcionamiento de la aplicación sea posible. La ventana de acceso al sistema de ficheros nos permite indicar un nombre y una ruta para el almacenamiento de ficheros de texto exportables.

Funcionamiento interno

Uno de los problemas más comunes en Java se debe a que el hilo de ejecución que se encarga del manejo de la interfaz es el mismo que el se encarga de la ejecución de cualquier tarea que lancemos. Este comportamiento provoca que, en caso de que la tarea asignada a la aplicación conlleve un tiempo de ejecución apreciable para el usuario, de la sensación de que la aplicación no responde, ya que en realidad mientras se ejecuta la tarea la interfaz queda congelada sin poder se usada en ese intervalo.

Evidentemente es un inconveniente muy grande que perjudica la implementación de barras de progreso para evitar exactamente lo que ocurre. Así que una solución eficiente que proporcione un buen refresco de la interfaz y además haga un mejor uso de la capacidad de computación del ordenador es el uso de hilos de ejecución paralelos.

Los hilos de ejecución en Java son muy fáciles de implementar y nos solventan problemas como los descritos arriba. Estos hilos pueden ser creados para la ejecución de una tarea en concreto siendo destruidos una vez se ha completado dicha tarea.

Se va a mostrar un ejemplo de código donde un hilo es lanzado para no provocar que la interfaz se congele mientras ejecuta una tarea de gran duración.

```
new Thread() {

    @Override
    public void run() {
        try {
            REACTION_BUTTON();
        } catch (InterruptedException ex) {
            Logger.getLogger(main_GUI.class.getName()).log(Level.SEVERE, null, ex);
        }
        EventQueue.invokeLater(new Runnable() {

            @Override
            public void run() {
                UpdateList();
                jButton1.setEnabled(true);
                jProgressBar1.setValue(0);
                jProgressBar1.setStringPainted(false);
                jComboBox1.setEnabled(true);
                jMenuItem2.setEnabled(true);
                jMenuItem4.setEnabled(true);
                jMenuItem5.setEnabled(true);
            }
        });
    }
}.start();
```

La creación de nuevo hilo vemos que se encarga de la ejecución que provoca el botón de búsqueda de reacciones mientras dejamos el hilo principal libre para el refresco de la interfaz.

Los hilos también tienen la capacidad de ejecutar funciones previas a su destrucción dentro de la cola de eventos gracias a su método “invokeLater”.

Uso de la aplicación y requisitos

La ejecución de la aplicación es posible en varios sistemas operativos como hemos comentado, todo ello gracias a la plataforma Java que permite su ejecución a través de su máquina virtual.

Como es evidente, la máquina virtual Java debe estar instalada en el ordenador para comenzar la ejecución de la aplicación; pero no es el único requisito indispensable para su funcionamiento.

Independientemente de los sistemas operativos disponibles, debemos tener ejecutándose un servidor MySQL cuya configuración es la siguiente:

User: root
Pass:
Host: localhost ó 127.0.0.1
Puerto: 3306

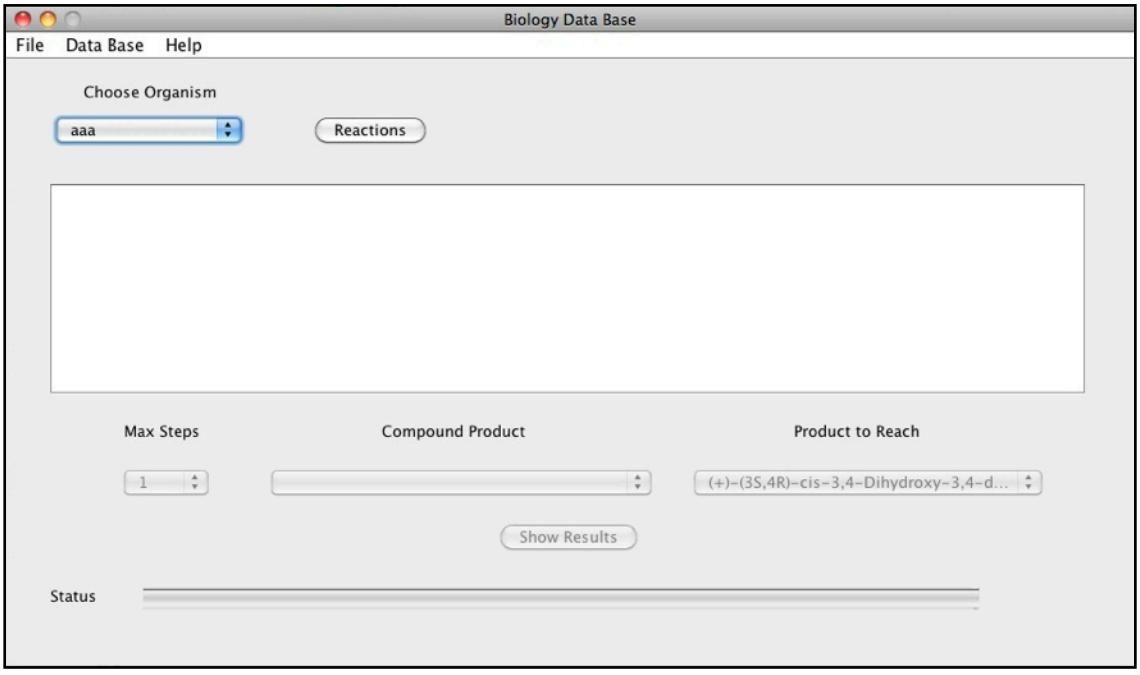
Una vez creado el servidor y éste se esté ejecutando en nuestro ordenador, disponemos de dos soluciones viables para la creación de la base de datos.

La primera opción, y tal vez la más viable, sea la importación de la base de datos que se adjunta con el proyecto y que tiene la base de datos actualizada a fecha de junio del 2011. Esta base de datos debe importarse con el manager de MySQL siguiendo sus instrucciones.

La segunda opción es hacerlo a través de la aplicación, descargando todos los ficheros del FTP de KEGG, cuyo tamaño estimado es de 6 GigaBytes y cuyo tiempo aproximado es de 5 días consecutivos teniendo en cuenta la conexión a internet que se tenga. La extracción de datos que se implementa en el programa también debe aplicarse tras la descarga y su construcción es de 10 horas aproximadamente dependiendo del procesador y la RAM del ordenador.

Aunque la segunda opción sea muy costosa, nos asegura tener la base de datos actualizada. Evidentemente se recomienda importar la facilitada en el proyecto y efectuar las acciones de descarga y extracción para actualizar la información ya existente. Los tiempos aproximados son mucho menores, siendo la descargada de 5 a 6 horas dependiendo de la cantidad de información a actualizar y de 3 a 4 horas la extracción dependiendo del mismo factor.

Teniendo la base de datos en orden ya podemos empezar a ejecutar acciones dentro de la aplicación. Hay que indicar que en el caso de no correr el servidor previamente, tras cierto número de intentos de conexión, la aplicación responderá con un mensaje de que no ha sido posible establecer contacto. Y en caso de que no haya ninguna tabla con datos dentro de la base de datos, las peticiones que se realicen devolverán mensajes de resultados vacíos.



Observando la ventana principal de la aplicación podremos comprobar que ninguna opción está habilitada, excepto construcción y extracción de la base de datos y la selección de un organismo para buscar sus reacciones asociadas. Por lo tanto, si ya hemos hecho el trabajo de crear la base de datos ahora, sólo debemos realizar búsquedas para estudiar los resultados que nos devuelva.

Una vez seleccionado el organismo del que queremos saber todas sus reacciones, pulsamos el botón “Reactions”, provocando que se deshabiliten todas las acciones hasta que la barra de progreso llegue al 100%.

Cuando termine, nos encontraremos que la lista muestra todas las reacciones. En este momento las funciones de la opción de la barra de herramientas “File” no da la opción de exportar todas las reacciones del organismo seleccionado en un archivo txt, mientras que la de exportar una sola reacción está deshabilitada hasta que seleccionemos una de la lista que se nos acaba de mostrar.

Al seleccionar una reacción concreta, no sólo se nos habilita la opción de “Export Selected Reaction” en la opción de la barra “File”, sino que también se nos habilita las listas desplegables de “Max Steps”, “Compound Product” y “Product to reach”. Las listas se habilitan para que junto con el botón “Show Results” podamos lanzar el algoritmo principal que se encarga de buscar las reacciones en cadena para alcanzar el producto seleccionado partiendo del producto de la reacción elegida.

Terminada la ejecución de la función de búsqueda llegando la barra de progreso al 100%, nos aparecerá una ventana en la que podremos mostrar los resultados de los organismos para los que se han encontrado resultados factibles, mostrándose así una lista de reacciones. También tendremos la posibilidad de exportar los resultados de un organismo seleccionado o de todos los organismos que hayan disponibles.

Todas las acciones que se han explicado pueden repetirse las veces que se requiera, teniendo en cuenta que los tiempos dependen de la capacidad de procesamiento del ordenador y que acciones como la búsqueda de reacciones en cadena dependen exponencialmente del factor “Max Steps”.

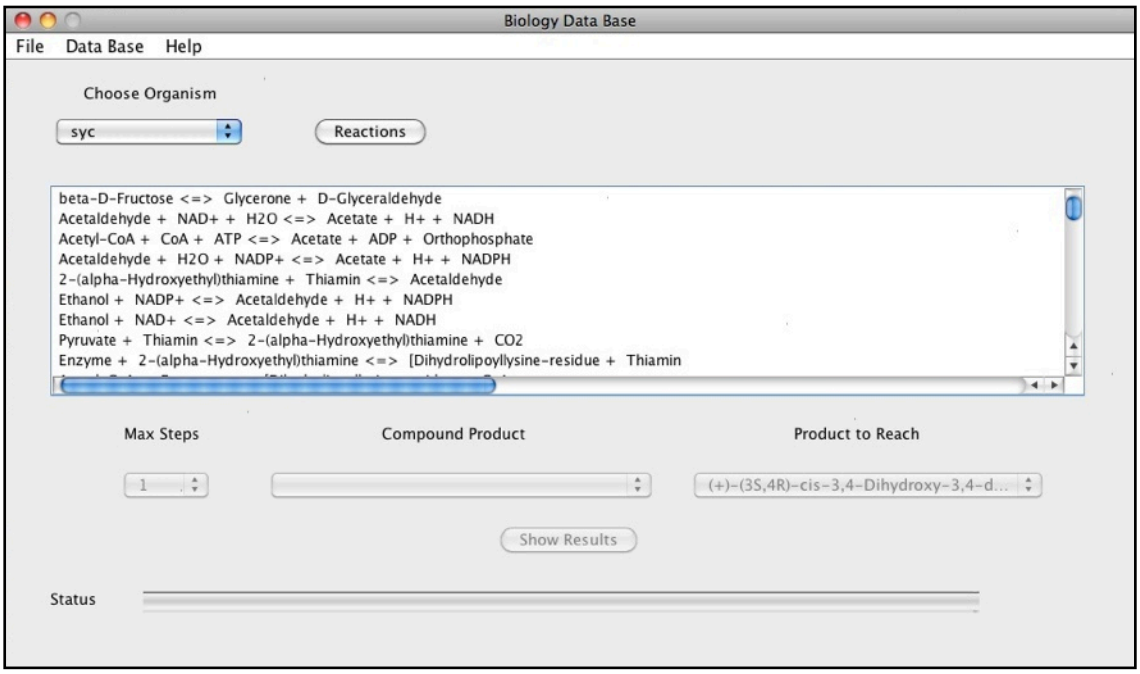
Capítulo 7
RESULTADOS

Terminada la versión final de la aplicación, es momento de realizar una serie de pruebas y observar si devuelve los resultados esperados y por lo tanto su función. Para ello se mostrarán ejemplos de tiempos y resultados aleatorios que se realizaron con el fin propuesto.

Búsqueda de reacciones

Como se ha comentado, la aplicación tiene la función de buscar en la base de datos todas aquellas reacciones asociadas a un mismo organismo. Así que haciendo uso de esta característica, se va a comentar los resultados devueltos después de su ejecución.

Si hacemos una búsqueda para el organismo "syc", observaremos que la lista se llena de reacciones hasta quedarse como en la figura de abajo.



La ejecución tarda unos 20 segundos aproximadamente para la mayoría de los casos y observar todas las reacciones es posible a través de la lista o exportándolas a través del menú "File".

Búsqueda de reacciones en cadena

Las reacciones en cadena tienen más variables para su búsqueda y los tiempo de ejecución difieren un poco más.

El primer ejemplo consiste en la búsqueda del componente "Glycocholate" partiendo del compuesto "CO2" de una reacción seleccionada. La variable de "Max Steps" se ha puesto a 8 con el fin de encontrar longitudes de encadenado para cubrir todo el rango.

El resultado alcanzado se muestra en formato de texto exportado:

```
aaa
-------
 Pyruvate +  Thiamin <=>  2-(alpha-Hydroxyethyl)thiamine +  CO2
 CO2 +  5-Hydroxykynurenamine <=>  5-Hydroxykynurenine
 5-Hydroxykynurenine +  cpd:C00161 <=>  4-(2-Amino-5-hydroxyphenyl)-2,4-dioxobutanoate +
cpd:C00045
 Leukotriene +  cpd:C00045 <=>  Leukotriene +  cpd:C03363
 H2O +  Leukotriene <=>  Glycine +  Leukotriene
 Glycine +  Choloyl-CoA <=>  CoA +  Glycocholate
```

En 5 pasos hemos encontrado una cadena que alcanza el compuesto objetivo dentro del organismo "aaa". El tiempo de ejecución es de aproximadamente 6 segundos.

Hay que tener en cuenta que hay muchos organismos que comparten pathways y por lo tanto es probable que si no hay un resultado concreto en uno de ellos, el primer organismo de la lista que contenga el pathway con la solución sea el seleccionado como en el caso del organismo "aaa".

El segundo y último ejemplo es la búsqueda de "Hydrogen" partiendo de "H2O". La variable de "Max Steps" es de 8.

El resultado que se obtiene tarda en ejecutarse mucho más que el anterior, en concreto 25 minutos más. Esto se debe a que estamos partiendo de un compuesto muy común como es el "H2O" y por lo tanto el árbol de soluciones se amplía mucho más.

Observamos 6 soluciones diferentes, unas alcanzadas en 1 paso y otras en 2 pasos. Cuál se elige como óptima depende de la energía que se consuma en la reacción y si compensa hacerlo en un número menor de pasos o no.

Sea como fuere la elección queda a criterio de biólogo que debe corroborar los resultados y aplicarlos en laboratorio.

aaa

2-Phospho-D-glycerate <=> Phosphoenolpyruvate + H2O
H2O + D-Cysteine <=> Pyruvate + NH3 + Hydrogen

2-Phospho-D-glycerate <=> Phosphoenolpyruvate + H2O
Oxygen + H2O + 3-Sulfocatechol <=> Sulfite + 2-Hydroxymuconate
L-Cysteine + Sulfite <=> Hydrogen + L-Cysteate

2-Phospho-D-glycerate <=> Phosphoenolpyruvate + H2O
Oxygen + H2O + Sulfur <=> Sulfite
L-Cysteine + Sulfite <=> Hydrogen + L-Cysteate

2-Phospho-D-glycerate <=> Phosphoenolpyruvate + H2O
H+ + H2O + ATP + Reduced + Nitrogen <=> ADP + Orthophosphate + NH3 + Hydrogen
+ Oxidized

2-Phospho-D-glycerate <=> Phosphoenolpyruvate + H2O
H2O + L-Homocysteine <=> NH3 + 2-Oxobutanoate + Hydrogen

2-Phospho-D-glycerate <=> Phosphoenolpyruvate + H2O
H2O + Diethylthiophosphoric <=> Hydrogen + Diethylphosphoric

Bibliografía

John Hunt, Alex McManus, “Key Java: advanced tips and techniques”, Springer-Verlag, London, ISBN 3-540-76254-0, (1998)

Stanley Letovsky, “Bioinformatics: databases and systems”, Springer, Massachusetts ISBN 0-7923-8573-X , (1999)

David W. Mount, “Bioinformatics: sequence and genome analysis”, Cold Spring Harbor, New York, ISBN 0-87969-687-7 ,(2004)

Baron Schwartz, Peter Zaitsev, Vadim Tkachenko, “High performance MySQL”, O´Reilly Media, Sebastopol, ISBN 0596101716, (2008)

Dave Minter, Jeff Linwood, “Pro Hibernate”, Springer-Verlag, New York, ISBN 1-59059-511-4 , (2005)

Johnny Brochard, “XML: conceptos e implementación” , Ediciones Software, Barcelona, ISBN 2-7460-1402-5 ,(2001)

Adam Myatt, Brian Leornard, Geertjan Wielenga, “Pro Netbeans IDE 6 Rich Client Platform Edition”, Springer-Verlag, New York, ISBN 1-59059-895-4, (2008)

Peter Rob, Carlos Coronel, “Sistemas de bases de datos: diseño, implementación y administración”, Thomson Ed., México DC, ISBN 0-619-06209-X , (2003)