Desarrollo de una plataforma de acceso a datos genómicos basada en Web

Directora de Proyecto: Mónica Chagoyen Quiles

Ingeniería Informática
Facultad de Informática

Curso Docente 2005 / 2006

Desarrolladores del proyecto:

- Beatriz Bravo Martínez.
- Luis Roldán López.
- Víctor Daniel Sánchez Pérez.

Autorización

Los siguientes alumnos, autores de este proyecto:

- *Beatriz Bravo Martínez* (DNI: 46854323-B)
- *Víctor Daniel Sánchez Pérez* (DNI: 52884201-W)
- *Luis Roldán López* (DNI: 05207031-S)

autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Beatriz Bravo Martínez

Víctor Daniel Sánchez Pérez

Luis Roldán López

Madrid, Julio de 2006

Índice

Resumen

Proyecto Genome es una plataforma de acceso a datos genómicos basada en Servicios Web que permite la obtención de datos de diferentes fuentes, permitiendo la colaboración entre diferentes grupos de investigación para la solución de problema biológicos importantes.

En concreto integra las siguientes bases de datos en una sola, actualizada diariamente: Gene Ontology (GO), Entrez Gene , Unigene, Saccharomyces cerevisiae (SGD), Mus musculus (MGI), Candida albicans (CGD), Rattus norvegicus (RGD) y Homo sapiens.

Proyecto Genome is a genomic data access platform based on Web Services to obtain data from different sources, permitting collaboration between differents investigation groups to solve some important biological problems.

Specifically the Proyecto Genome database, updated daily, consists of these databases: Gene Ontology (GO), Entrez Gene , Unigene, Saccharomyces cerevisiae (SGD), Mus musculus (MGI), Candida albicans (CGD), Rattus norvegicus (RGD) and Homo sapiens.

1. Introducción

1.1 Motivación y situación actual

Desde hace unos años, las fuentes de que dispone la biología molecular para la investigación se han visto desbordadas por un crecimiento desproporcionado de su volumen de información. Esta cantidad de datos no puede seguir siendo publicada de la forma tradicional, es decir, en revistas y publicaciones científicas, sino que se necesitan otras formas de almacenamiento que lleguen a toda la comunidad científica. Es aquí donde nace la bioinformática, que, como disciplina científica que utiliza la informática para organizar, analizar y distribuir información biológica, tiene como principal objetivo facilitar la investigación con la creación y mantenimiento de bases de datos.

Así por ejemplo, gracias a la bioinformática puede leerse toda la secuencia de moléculas de un organismo, es decir, el "manual de instrucciones" que rige la formación del mismo y es posible la observación de su comportamiento dinámico bajo diversas condiciones experimentales.

Entre todas las bases de datos surgidas para unificar información genética cabe destacar la del NCBI (Nacional Center for Biotechnology Information), responsable de la creación y el mantenimiento de GenBank, una base de datos que reúne todo el conocimiento sobre las secuencias de ADN, **Gene Ontology**, principal referente para la anotación de genes y productos genéticos o, en el ámbito bibliográfico, **PubMed**, una base de datos de información bibliográfica extraída principalmente de literatura relativa a las ciencias naturales.

En el caso concreto de las bases de datos genómicos, aparece el problema de la falta de compatibilidad entre ellas. Las herramientas disponibles para la consulta y la navegación por dichas bases de datos no permiten la obtención de datos de diferentes fuentes, repercutiendo esto en la colaboración y la cooperación entre distintos grupos de investigación en la solución de problemas biológicos importantes.

Podrían resumirse en cuatro las dificultades existentes hoy en día en el acceso a datos biológicos/genómicos:

- Bases de datos distribuidas
- Heterogeneidad tanto sintáctica como semántica (formato, esquemas, etc.)
- Expresividad limitada en las interfaces de consulta
- Crecimiento exponencial de los datos acumulados

1.2 Objetivos

Teniendo en cuenta la situación actual descrita anteriormente se propone el Proyecto Genome, una plataforma de acceso a datos genómicos basada en servicios web, permitiendo así el uso entre distintas aplicaciones bioinformáticas. En particular, se propone integrar el acceso a la información funcional de los distintos genomas anotados con el vocabulario control definido en Gene Ontology.

El uso de servicios web ofrece una capa de abstracción a alto nivel que ocultaría los detalles de implementación de los accesos a las distintas fuentes de datos que cada grupo de investigación necesita consultar. De esta manera se puede implementar una plataforma de acceso a datos "unificada" a la vez que se proporciona una interfaz programable.

Los objetivos concretos del proyecto son:

1. Desarrollar un sistema, basado en servicios web, para el acceso a las anotaciones funcionales de los genomas incluidos en el consorcio Gene Ontology, así como los proporcionados por la base de datos Entrez Gene.

2. Desarrollo de los servicios que respondan a las consultas más frecuentes, definidas en el documento de requisitos del sistema)

3. Proporcionar los mecanismos de actualización necesarios para garantizar el acceso a las últimas versiones de los datos disponibles

4. Implementación de un prototipo web para la demostración interactiva de los servicios proporcionados.

1.3 Organización de esta memoria

Este documento está estructurado del siguiente modo:

- **Capítulo 1 - Introducción, objetivos y motivación de este proyecto**: situación en el ámbito de la bioinformática y el contexto en que se desarrolla este proyecto. Explicación de las causas por las cuales se necesitó y se decidió llevar a cabo y cuales son las necesidades que pretende cubrir.

- **Capítulo 2 - Proyecto GO**: breve introducción y explicación de la estructura y objetivos del Proyecto Gene Ontology, principal referente en el ámbito de la bioinformática de la actualidad, sobre el cual se basa el Proyecto Genome.

- **Capítulo 3 - Tecnologías y herramientas utilizadas**: explicación a nivel teórico de cuales han sido las tecnologías empleadas en el desarrollo de este proyecto, tanto en lo referente al diseño de la herramienta web que permite el uso de todo el código desarrollado como en lo referente a la implementación del propio código.

- **Capítulo 4 - Diseño e implementación**: esclarecimiento de ciertos detalles de la implementación sin los que no se puede entender correctamente el sistema de venta de entradas, desde un punto de vista técnico. Abordaremos en este apartado el diseño de la herramienta web y de la base de datos.

- **Capítulo 5 - Manual de usuario**: manual que, a modo de instrucciones, pretende orientar al usuario a la hora de usar por primera vez la herramienta web, explicando de forma detallada que consultas permite realizar y que datos puede introducir y obtener.

- **Conclusiones**

- **Bibliografía**

- **Glosario**

2. El proyecto Gene Ontology

2.1 Introducción: ¿Qué es una ontología?

Podemos encontrar diferentes definiciones para la palabra "Ontología". La definición oficial de la **Real Academia Española** es "*Parte de la metafísica que trata del ser en general y de sus propiedades trascendentales*". Sin duda, esta definición no nos sirve de mucha ayuda desde el punto de vista biológico, pero veamos a continuación otras definiciones que se acercan más a lo que queremos explicar:

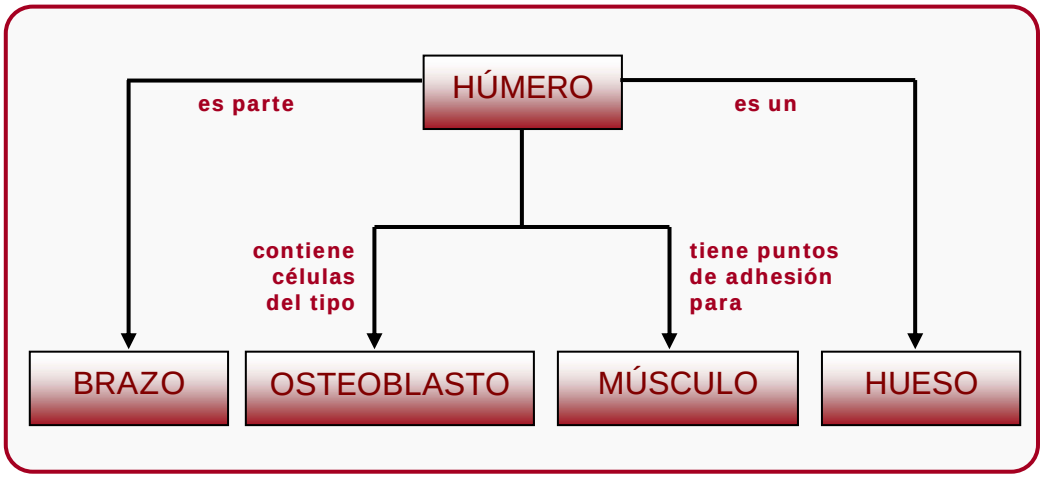
- *"Parte de la ciencia de lo metafísico que investiga y explica la naturaleza, las propiedades esenciales y las relaciones de todos los seres."*

- *"Estructuración jerárquica del conocimiento acerca de las cosas, mediante la clasificación en categorías de acuerdo con sus propiedades esenciales (o al menos relevantes y/o cognitivas)."*

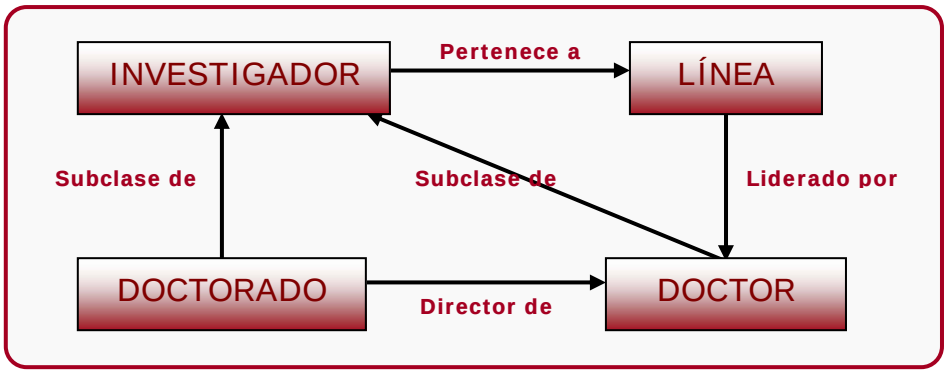
- *"Área del conocimiento que ha sido formalizada"*

Una vez aclarada la definición de ontología, presentamos a continuación algunos ejemplos ilustrar dicha definición:

- **ONTOLOGÍA:** Ontología anatómica
- **TÉRMINOS:** Músculo, tejido, hueso, fémur, músculo…
- **RELACIONES:** es parte de, es un, contiene células del tipo, es un punto de adhesión de, etc**.**



- **ONTOLOGÍA:** Ontología de investigación
- **TÉRMINOS:** Doctorado, doctor, investigador, línea…
- **RELACIONES:** pertenece a, esta liderado por, es director de, es un, etc.



2.2 Historia de las ontologías biológicas

Las primeras formas de clasificación del conocimiento nos llevan al siglo IV d.C. con Aristóteles y su obra “Scalae naturae of animals”. Theophrastus, sucesor de Aristóteles como líder de su escuela en Atenas, constituye la aportación más importante desde la antigüedad hasta la Edad Media en el campo de la botánica con sus obras “Historia de las plantas” (nueve libros, aunque diez originalmente) y “Tratado sobre las Causas de las Plantas”.

Ya en el siglo XVIII nos encontramos con Carl Linnaeus, creador del sistema de clasificación de plantas que lleva su nombre. Linnaeus dedica 23 clases al tratamiento de las fanerógamas, agrupando todas las plantas sin flores en una última clase, que propone a modo de cajón de sastre. Linnaeus propuso un nuevo método de clasificación de plantas basado en el sistema binomial: cada planta se distingue por un doble término, un sustantivo que indica el género y un adjetivo que indica la especie. De esta forma todas las plantas de una misma especia natural comparten el mismo género. El sistema funcionó tan bien que se ha mantenido hasta la actualidad.

2.2.1 Algunas ontologías biológicas

En más de una ocasión en la historia de la bioinformática se han intentado crear ontologías que integraran todos los conceptos biológicos, pero casi siempre ha fallado el apoyo popular. Entre los casos exitosos encontramos los típico sistema de keywords y features de diversas bases de datos (UniProt, GenBank, EMBL, PDB, etc...) o la anotación mediante códigos de actividad enzimática (p.ej., base de datos ENZYME), sin equivalencias posibles entre los conceptos usados para anotar en las distintas bases de datos.

En la actualidad existen tres tipos de proyectos relacionados con representación de conocimiento en bioinformática, que son:

1. Bases de datos relacionales
2. Taxonomías sin o con muy poco conocimiento axiomático
3. Sistemas que intentan simular el razonamiento humano en dominios muy específicos

Como ejemplo del primer grupo de proyectos, que será el que tratemos en este proyecto, tenemos Gene Ontology, que nació para poder unificar todos los conceptos biológicos existentes, usando además estándares existentes como RDF para la representación de los conceptos y sus interrelaciones. Actualmente podemos encontrar bases de datos como InterPro, que anotan sus contenidos usando los conceptos de Gene Ontology.

2.2.2 El Proyecto OBO

El proyecto OBO (Open Biomedical Ontologies) está diseñado por el nacional Center for Biomedical Ontology como un sitio donde almacenar y compartir ontologías de los diferentes dominios médicos y biológicos y que estén disponibles para todo la comunidad científica interesada en ellas. La ontología más conocida de todas las que conforman OBO es la génica, en la que los genes se clasifican en función de la actividad principal o función del producto genético, el lugar donde un proceso biológico ejerce su función y los procesos biológicos a los que esta vinculados. Esta ontología, que veremos mas detalladamente en el siguiente apartado, aportó muchas ventajas a la investigación genética.

Desde la página web de OBO, http://obo.sourceforge.net/, se puede acceder a todas las ontologías almacenadas mediante un navegador que las muestra en relación a su categoría. Con intención de acercarse a toda la comunidad, OBO acepta ontologías presentadas por investigadores interesados en la materia. Para que una ontología entre a formar parte del proyecto OBO debe cumplir los siguientes requisitos:

1. Ser de fuente abierta y poder ser utilizada sin restricción alguna.
2. Utilizar una sintaxis común.
3. No solaparse ni competir con ninguna otra antología.
4. Tener identificadores y definiciones únicas.

2.3 El proyecto Gene Ontology

2.3.1 Introducción

El proyecto Gene Ontology, también llamado Proyecto GO, proporciona vocabularios y clasificaciones estructuradas y controladas que abarcan muchos dominios en la biología molecular y celular, y que están disponibles de forma gratuita para su uso en el trabajo con genes.

Son muchas las bases de datos de modelos de organismos y anotaciones genéticas que utilizan el proyecto Gene Ontology, aportando a éste nueva información. La base de datos de GO contiene todas estas ontologías, vocabularios y anotaciones, permitiendo el acceso a dichos datos en diferentes formatos.

Los miembros del GO Consortium trabajan continuamente para ampliar y mantener actualizada la nomenclatura de GO, poniendo a disposición de cualquiera una página Web donde se pueden encontrar tantos los datos como links a aplicaciones que utilizan dicho material para realizar análisis funcionales.

2.3.2 El nacimiento del proyecto Gene Ontology

En la era de la biología genómica se ha generado una acumulación de datos biológicos, acompañada de una proliferación de bases de datos destinadas a utilizarlos. Para hacer un mejor uso de toda esta información, se buscan formas de integrar las diferentes fuentes de que se dispone para ponérselo más fácil a los biólogos.

Un componente muy importante de todo este proceso de integración es el uso de ontologías, ya que ofrecen la estandarización de los conceptos para permitir una mejor comunicación entre investigadores y herramientas informáticas.

El proyecto Gene Ontology nace en 1998 de mano de un consorcio de investigadores que estudian el genoma de organismos tales como Droshopila (la mosca de la fruta), RATON y Saccharomyces (levadura). Se comenzó a crear un vocabulario estándar para describir las funciones de dichos genes y se estructura dicho vocabulario en forma de ontología, una estructura, como dijimos antes, en la que cada término está enlazado con otros términos por medio de una serie de relaciones.

Desde su creación, Gene Ontology ha crecido absorbiendo a más de 15 bases de datos de ámbito genético, con más de un millón de productos genéticos clasificados en aproximadamente 17.000 términos. Hoy en día contiene algunas de las más importantes fuentes de datos en el campo de los genes, las plantas y los animales, convirtiéndose en el principal referente en el ámbito de la bioinformática.

El proyecto Gene Ontology, surgió por tanto con tres fines principales:

- Desarrollar una serie de vocabularios estructurados y estandarizados (ontologías) para describir dominios de biología molecular, características de los elementos genéticos y secuencias biológicas

- Aplicar los términos creados por GO en la anotación de secuencias, genes o productos genéticos en las bases de datos biológicas

- Proporcionar una fuente de datos a la que tuvieran acceso las ontologías, las notaciones y las herramientas informáticas.

2.3.3 Estructura de GO

Gene Ontology se concibe como una ontología global que abarca otras tres ontologías generales, que pueden ser utilizadas independientemente y que describen atributos de elementos genéticos de cada uno de estos dominios de biología molecular.

Estos tres subconjuntos son:

- ***FUNCION MOLECULAR (Molecular function):*** las anotaciones de este subconjunto se refieren a la actividad principal o función del producto genético de un gen, independientemente de donde y cuando realiza dicha función.

  Algunos ejemplos de los niveles superiores de la jerarquía serían "enzima" o "proteína estructural", mientras que "adenilato ciclasa" o "actina" serían ejemplos de niveles inferiores y, por tanto, funciones más específicas.

- ***PROCESO BIOLÓGICO (Biological proccess):*** las anotaciones de esta ontología hacen referencia a la dinámica en que está englobada una proteína, procesos que son llevados a cabo por conjuntos ordenados de funciones moleculares. Así pueden definir rutas metabólicas o procesos celulares.

  Algunos ejemplos serían "mitosis" o "metabolismo de purinas".

- ***COMPONENTE CELULAR (Cellular component):*** Describe estructuras subcelulares, localizaciones, complejos macromoleculares… en definitiva, lugares donde un proceso biológico ejerce su función. Así podríamos encontrar la anotación "citoplasma", dentro de ella "mitocondria" y con mayor detalle "membrana mitocondrial interna"

  Otros ejemplos de componentes celulares podrían ser "núcleo" o "telomero".

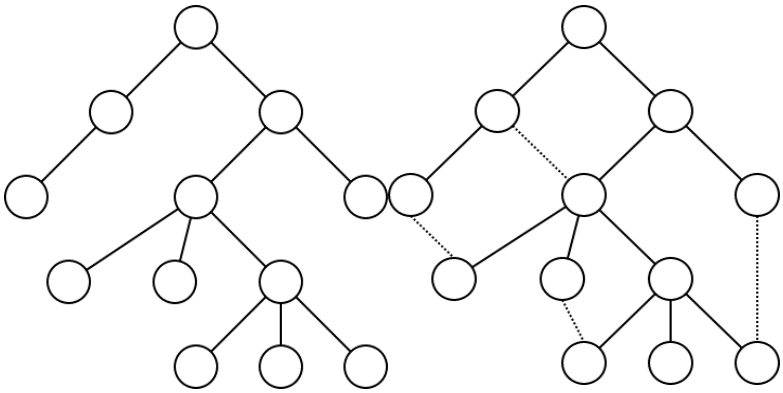
Es importante destacar que cualquier gen puede ser mapeado en estas ontologías, es decir, el producto de un gen determinado tiene una **función molecular**, es parte de algún

proceso biológico y ocurre en algún **componente celular**. Dentro de cada ontología, los términos tienen un identificador único.

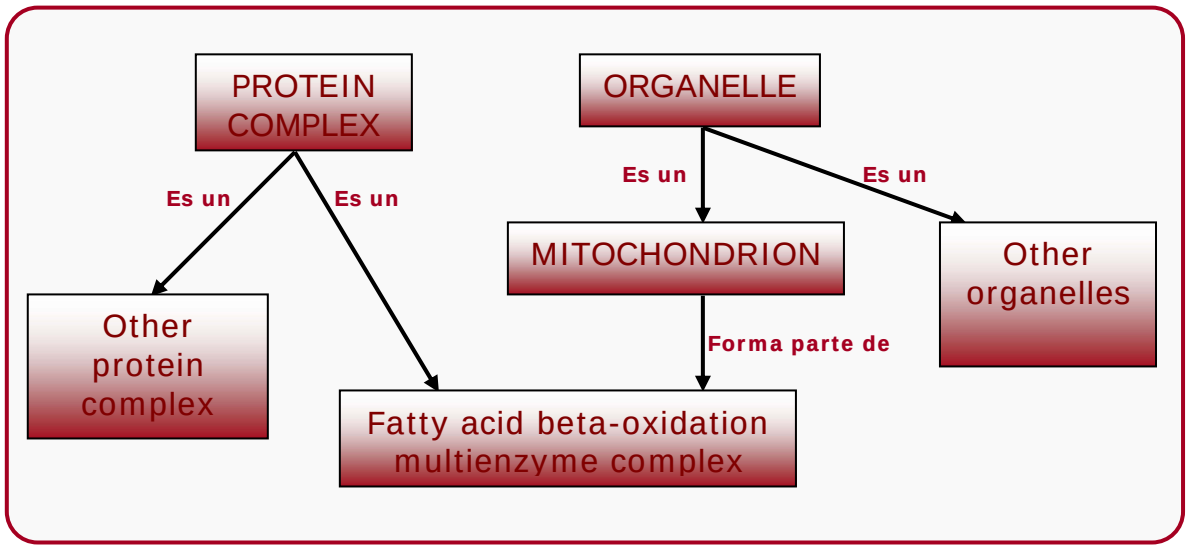
En la actualidad, dado que la investigación genética genera nuevos resultados muy rápidamente, es fundamental la existencia de estas tres ontologías rigurosamente definidas.

La característica más importante de la estructura de Gene Ontology permite que sea entendido como un grafo acíclico dirigido, donde los nodos serían los términos GO y las aristas serían las relaciones entre los nodos. Cada nodo puede tener más de un padre y ninguno o más hijos. Las relaciones definidas por las aristas son del tipo: "es un", "forma parte de", etc.

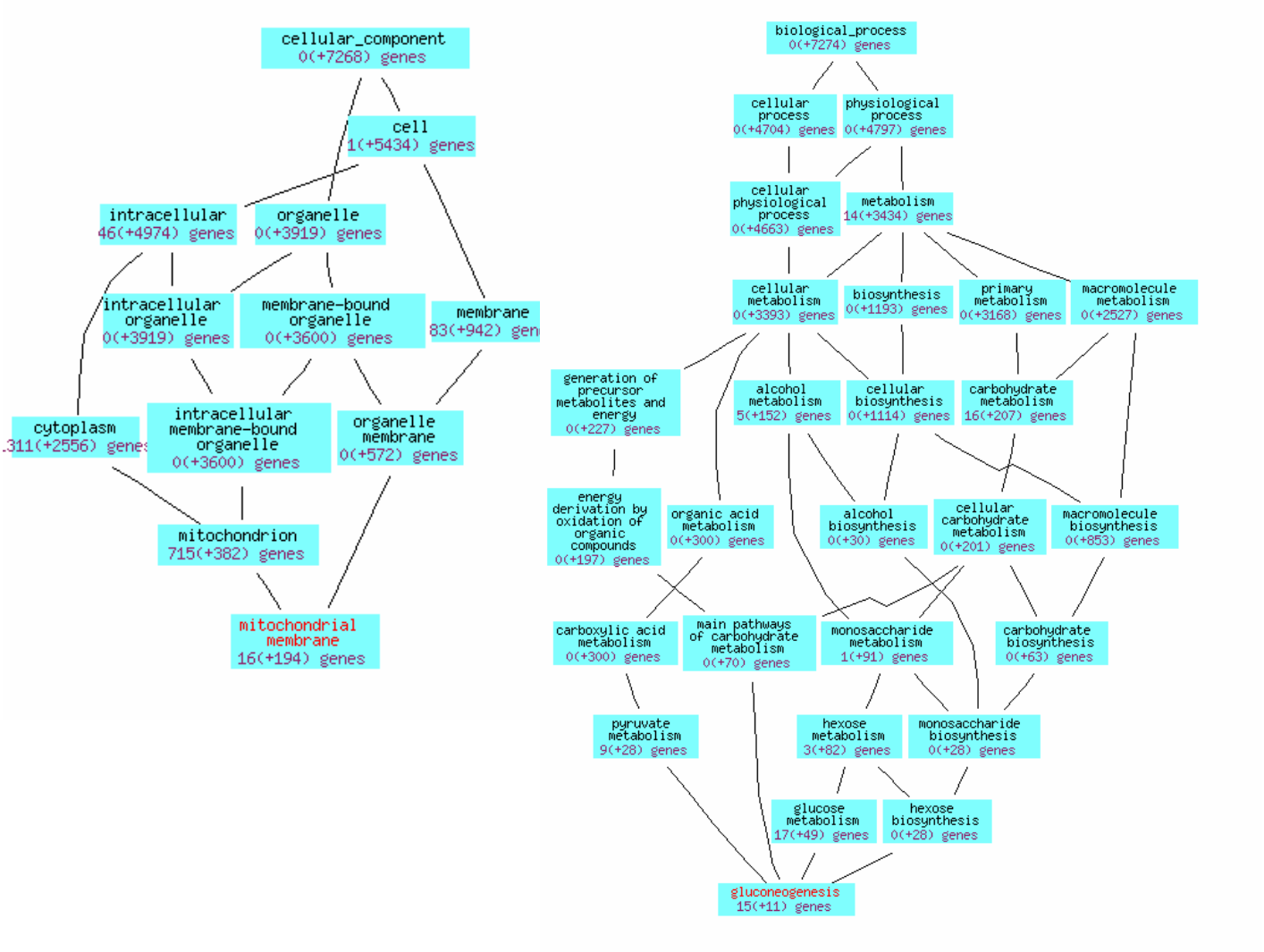
ÁRBOL **GRAFO ACÍCLICO DIRIGIDO**



Ejemplo:



Los grafos pueden llegar a ser de tamaños considerables, como se muestra en los siguientes ejemplos para “Membrana mitocondrial” y “Gluconeogenesis”.



2.3.4 Términos GO

Un término GO describe una función molecular, un proceso biológico o un componente celular, pero no representa ni un gen ni un producto genético. Cada término del vocabulario de Gene Ontology consiste en un identificador alfanumérico único, un nombre común y una definición. Los términos se clasifican en una de las tres ontologías descritas: función molecular, proceso biológico y componente celular.

Así, un término de GO está definido por tres elementos:

1. **Nombre** (Ej: “*gluconeogenesis*”)
2. **Identificador único** (Ej: “*GO:0006094*”)
3. **Definición** (Ej: “*The formation of glucose from noncarbohydrate precursors, such as pyruvate, amino acids and glycerol.*”)

Actualmente la base de datos cuenta con:

Función molecular……7.309 términos
Proceso Biológico……10.041 términos
Componente celular……1.629 términos

TOTAL……18.975 términos

2.3.5 Anotaciones

Las anotaciones del proyecto Gene Ontology consisten en relaciones entre los productos genéticos y los términos de GO que los describen. El uso de los términos GO en las bases de datos de anotaciones es quizá el aspecto más importante de Gene Ontology, ya que la anotación de productos era el principal objetivo de este proyecto.

En Gene Ontology, la anotación de un producto genético puede representarse como una tupla que contiene la información de la **función molecular** que desempeña, **el proceso biológico** de que forma parte y si pertenece a algún **componente celular**. Por ejemplo:

- **Organismo**: Mitocondria P450
- **Componente celular***:* membrana mitocondrial interna (GO:0005743)
- **Proceso biológico:** transporte de electrones (GO:0006118)
- **Función molecular:** proceso biológico catalizado por la monooxigenasa (GO:0004497)

Cada anotación debe indicar además el tipo de prueba que aporta y debe ser atribuida a una fuente, que puede ser de uno de los siguientes tipos:

1. Referencia literaria
2. Otra base de datos
3. Análisis computacional

Si la anotación es inferida por algún tipo de programa informático llevará el código IEA (Inferred from Automatic Annotation). Cualquier código distinto de ese implica que ha habido una comprobación “humana” de esos datos. En el siguiente gráfico se muestran todas las posibles fuentes:

Código	Definición
IEA	**I**nferred from **E**lectronic **A**nnotation
NAS	**N**on-traceable **A**uthor **S**tatement
TAS	**T**raceable **A**uthor **S**tatement
ND	**N**o **D**ata
IDA	**I**nferred from **D**irect **A**ssay
*IPI	**I**nferred from **P**hysical **I**nteraction
*IGI	**I**nferred from **G**enetic **I**nteraction
IMP	**I**nferred from **M**utant **P**henotype
IEP	**I**nferred from **E**xpression **P**attern
*IC	**I**nferred from **C**urator
*ISS	**I**nferred from **S**equence **S**imilarity

En la página de Gene Ontology podemos encontrar las tablas de anotaciones de más de 30 organismos diferentes, aunque en este proyecto nos hemos centrado en los siguientes:

1. **Saccharomyces cerevisiae (SGD)**
2. **Mus musculus (MGI)**
3. **Candida albicans (CGD)**
4. **Rattus norvegicus (RGD)**
5. **Homo sapiens**

Más adelante veremos en detalle que aspecto tienen dichas tablas.

2.3.6 GO-Slims

En muchas ocasiones es muy útil contar con una “visión de alto nivel” de cada una de las tres ontologías anteriormente descritas (función molecular, proceso biológico y componente celular) de forma que podamos describir con menos categorías las funciones de los genes. Estos subconjuntos de términos de GO que proporcionan esa visión general se conocen como “GO slims”. El primer Go Slim se construyó para la anotación del genoma de la mosca Drosophila.

Un GO Slim es por tanto, una selección de los términos de las ontologías proceso biológico, función molecular y componente celular que representan la mayoría de las ramas de cada ontología (recordemos que una ontología puede verse como un árbol). En general se trata de nodos del grafo relativamente cercanos a la raíz.

Por ejemplo, el término “núcleo” es un GO Slim para la ontología componente celular. Sus hijos (espacio perinuclear, matriz nuclear, etc.) son términos más detallados de GO y no forman parte del subconjunto del GO Slim.

Estas listas de términos son creadas por los usuarios de acuerdo a sus necesidades, y pueden ser específicas a especies o áreas particulares de las ontologías. El proyecto GO proporciona varios GO slims predeterminados, que se actualizan continuamente:

Nombre	Autor
Generic GO slim	Suparna Mundodi and Amelia Ireland
GOA and whole proteome analysis	N. Mulder, M. Pruess
Plant GO slim	Suparna Mundodi
Yeast GO slim	SGD curators

Además podemos encontrar un serie de GO Slims que han quedado obsoletos y ya no se actualizan. Las razones por la que aún siguen disponibles son dos: la necesidad de ver que términos se utilizaron para obtener los resultados de cierto análisis y reutilizar esas listas de términos para generar otras nuevas.

En la siguiente tabla aparecen todos ellos:

Nombre	Autor(es) / Publicación	Fecha
generic.0208	Suparna Mundodi and Amelia Ireland	Ago 2002
Apis_EST.0402	Whitfield CW, Band MR, Bonaldo MF, Kumar CG, Liu L, Pardinas JR, Robertson HM, Soares MB, Robinson GE.	Apr 2002
Drosophila.0200	M. Adams, M. Ashburner, G.M. Rubin, S.E. Lewis et al.; Adams et al. 2000.	Feb 2002
Glossina_EST.0905	M. Berriman	Sep 2002
goa.2002	N.Mulder, M.Pruess	Nov 2002
Mouse_Riken.0201	The RIKEN Genome Exploration Group Phase II Team and the FANTOM Consortium	Feb 2001
Pfalciparum.2002	M. Berriman	Jul 2002
plant.2003	Suparna Mundodi	Dec 2002
Rice_Beijing.0204	Yu, J., et al.	Apr 2002
Rice_Syngenta.0204	Yu, J., et al.	Apr 2002
yeast.2003	SGD curators	Ago 2003

Todos estos archivos están disponibles tanto en formato OBO como en el antiguo formato GO, en la url: http://www.geneontology.org/GO.slims.shtml

2.3.7 La base de datos Gene Ontology

La base de datos del proyecto GO se construye directamente a partir de los ficheros que contienen toda la información de las tres ontologías y de las anotaciones de los distintos organismos. Esta base de datos relacional que está disponible de forma gratuita en la página Web del proyecto.

Posteriormente, en el apartado de diseño, veremos en detalle como está estructurada esta base de datos

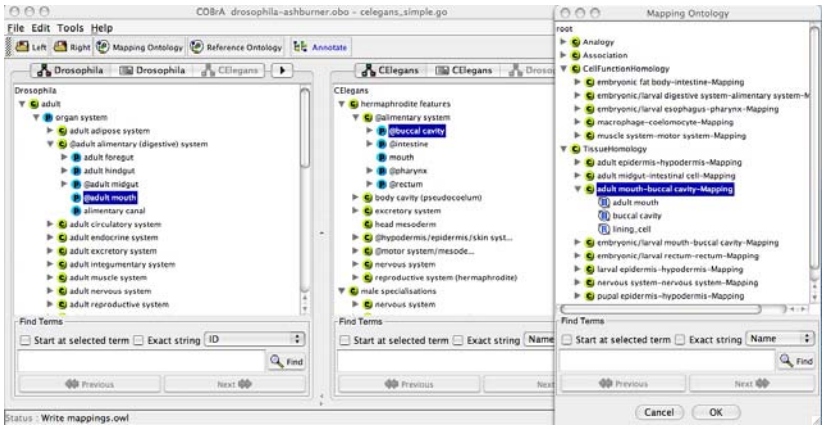
2.3.8 Herramientas para la consulta de Gene Ontology

Existen numerosas herramientas para la consulta de la base de datos de GO y para realizar análisis funcionales basados en los datos de la misma. A continuación presentamos brevemente algunas de las más conocidas:

EDITORES:

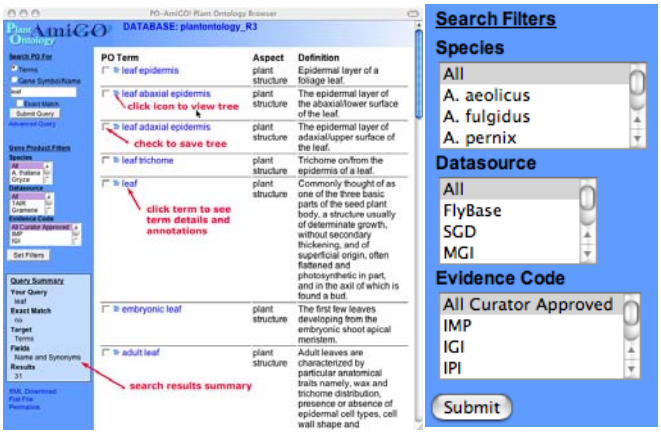
1. **DAG-Edit**: se trata de una herramienta con una interfaz gráfico para consultar, ejecutar querys sobre la base de datos y editar texto de cualquier vocabulario con una estructura similar a la de GO. Esta herramienta ha sido utilizada por numerosos grupos de biólogos para construir ontologías de un gran número de especies y organismos. Se trata de una herramienta de código abierto, escrita en JAVA y que está disponible de forma gratuita en la siguiente dirección: http://www.geneontology.org/doc/dagedit_userguide/dagedit.html.

2. **COBra**: este editor implementado en JAVA destaca por soportar la relación de conceptos entre dos ontologías diferentes, dando la opción al usuario de realizar por tanto análisis muy complejos. Permite importar y exportar ontologías en los formatos semánticos RDF, RDFS y OWL, además de los formatos GO (Gene Ontology) y OBO (Open Biology Ontologies). Está disponible en la siguiente dirección: http://www.xspan.org/cobra/.

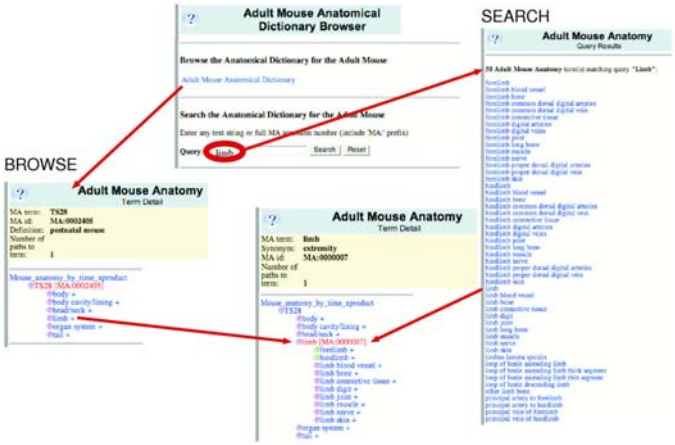


NAVEGADORES, HERRAMIENTAS DE CONSULTA:

1. **amiGO**: se trata de una aplicación web que permite consultar, ejecutar querys y visualizar el contenido de la base de datos de Gene Ontology o cualquier ontología en formato OBO. Está escrita en JAVA, es de código abierto y puede ser descargada en la siguiente url: http://www.godatabase.org/cgi-bin/amigo/go.cgi



2. **MGI GO Browser**: esta herramienta de consulta permite al usuario acceder al contenido de la ontología del ratón (MGI). En la consulta de un determinado término se puede obtener un gráfico que representa los padres e hijos de dicho término, así como un enlace a todas las notaciones de ese término o cualquiera de sus subtérminos.

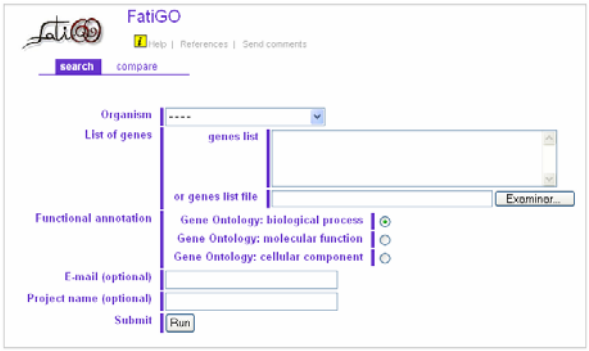


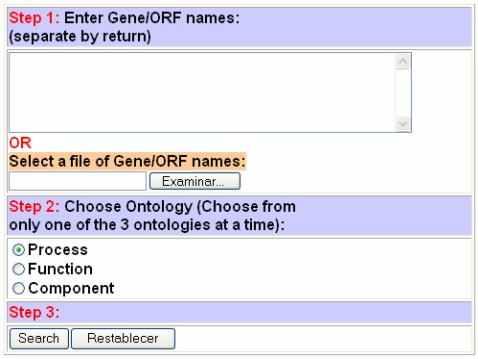
3. **GO Fish**: disponible como un applet de JAVA, permite al usuario construir querys utilizando términos GO y estimar, para cada producto genético, la probabilidad de que satisfagan dicha query. Disponible en la dirección: http://llama.med.harvard.edu/~berriz/GoFishWelcome.html

OTRAS HERRAMIENTAS:

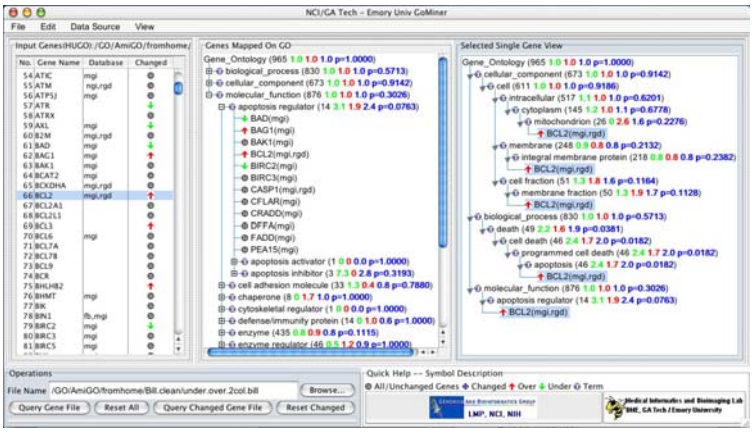
1. **SGD Term Finder**: consiste en una herramienta que permite buscar, en la base de datos de la levadura (SGD), aquellos términos GO (o sus padres), que son compartidos por un grupo de genes introducidos por el usuario. Disponible en la url: http://db.yeastgenome.org/cgi-bin/GO/goTermFinder

2. **FatiGO**: se trata de una aplicación web que permite ejecutar querys sobre la base de datos GO y que permite encontrar relaciones significativas de términos GO con grupos de genes. Se trata de una aplicación bastante restrictiva, en tanto que el usuario debe especificar el nivel concreto de la jerarquía de GO en que se debe realizar el análisis. Disponible en la url: http://fatigo.bioinfo.cnio.es





3. **GOMiner**: una herramienta escrita en JAVA empleada para la interpretación de datos genómicos obtenidos de experimentos. Clasifica los genes en categorías biológicas utilizando la ontología GO y las evalúa estadísticamente, proporcionando como salida una estructura arbórea similar a la de la herramienta amiGO. Se puede encontrar en la siguiente dirección: http://discover.nci.nih.gov/gominer/



Existen numerosas herraminetas, paquetes y proyectos desarrollados para el uso y la explotación de la base de datos de Gene Ontology. Todas ellas están disponibles en la página http://www.geneontology.org/GO.tools.shtml del proyecto Gene Ontology.

2.3.9 La página web de Gene Ontology

En la página web del proyecto GO http://www.geneontology.org están disponibles todas las versiones de las ontologías y las definiciones de los términos GO, así como las anotaciones de los distintos organismos.

Hay que destacar que los ficheros de las ontologías pueden encontrarse en cuatro formatos diferentes: el formato OBO (Open Biomedical Ontologies), en el formato GO, aunque este ya ha quedado en desuso, en formato XML y en OWL. Además se pueden descargar en formato SQL, para construir directamente la base de datos.

Por su parte las anotaciones están almacenadas en unos ficheros tabulares especialmente creados por el GO Consortium. La descripción detallada de cada campo de estos ficheros esta disponible en: http://www.geneontology.org/GO.annotation.shtml#file

En la página web de GO se puede encontrar toda la información referente a la base de datos del proyecto, así como artículos, resultados de experimentos y enlaces a todas las herramientas que hay disponibles para el uso de GO.

3. Tecnologías empleadas

3.1 Tecnologías y Herramientas

3.1.1 Java

Java es un lenguaje de programación de propósito general, orientado a objetos, que fue introducido por Sun Microsystems en 1995, y diseñado en principio para el ambiente distribuido de Internet. Pero lo que hace de Java un concepto diferente es que es también un entorno para la ejecución de programas, englobado en la llamada máquina virtual de Java. Este entorno es un software que permite que las aplicaciones escritas en Java se ejecuten en cualquier ordenador, independientemente del sistema operativo y de la configuración de hardware utilizados.

Las principales características de Java son:

- **Universalidad.** La universalidad de los byte codes hacen de Java el lenguaje idóneo para desarrollar aplicaciones para Internet. De hecho, la mayor parte de los navegadores (Netscape Navigator, Internet Explorer, HotJava) integran máquinas virtuales, y por tanto intérpretes de Java, lo que hace posible acceder automáticamente a los applets presentes en las páginas web. La sencillez de Java hace que esta integración no reduzca en absoluto las prestaciones de los navegadores, permitiendo además la **ejecución rápida y simultánea** de gran cantidad de applets.

- **Sencillez.** Java es un lenguaje de gran facilidad de aprendizaje, pues en su concepción se eliminaron todos aquellos elementos que no se consideraron absolutamente necesarios. Por ejemplo, en comparación con otros lenguajes como C ó C++, es notable la ausencia de punteros, o lo que es lo mismo: es imposible hacer referencia de forma explícita a una posición de memoria; ello ahorra gran cantidad de tiempo a los programadores, dado que el comportamiento imprevisto de los punteros es una de las principales fuentes de errores en la ejecución de un programa. Por otra parte, el código escrito en Java es por lo general mucho más legible que el escrito en C ó C++.

- **Orientación a objetos.** Aunque C++ es también, como Java, un lenguaje orientado a objetos, la diferencia fundamental es que Java lo es desde su concepción, mientras que C++ se diseñó como un lenguaje compatible con C (que no es orientado a objetos): de este modo, un programa escrito en C++ puede ignorar la mayoría de las características de orientación a objetos de C++ y compilarse sin problemas en un compilador de C++. Sin embargo, un programador no puede obviar la orientación a objetos cuando escribe un programa en Java, y esto hace que las aplicaciones escritas en Java tengan interesantes ventajas.

- **Seguridad.** En general, se considera que un lenguaje es tanto más seguro cuanto menor es la posibilidad de que errores en la programación, o diseños malintencionados de programas (virus), causen daños en el sistema. La extrema seguridad de Java se establece a tres niveles:

  - Nivel de seguridad *dado por las características del lenguaje*, tales como la ausencia de punteros (que evita cualquier error de asignación de memoria) o el "ocultamiento de la información" propio de la programación orientada a objetos, por recordar dos ejemplos ya mencionados.

  - Nivel de seguridad *dado por el diseño de la VM*. La VM de Java posee un **verificador de los byte codes**, que antes de ejecutarlos analiza su formato comprobando que no existen punteros en ellos, que se accede a los recursos del sistema a través de objetos de Java, etc. Otro elemento constitutivo de la VM es el **cargador de clases**. Una clase es una categoría de objetos utilizados en un programa; cuando se ejecuta un programa en Java, éste llama a determinadas clases a través del cargador de clases. Estas clases pueden provenir de tres lugares distintos, en donde residen en forma de ficheros: del ordenador local, de la red de área local a la que pueda estar conectado el ordenador cliente, o de Internet. En función de la procedencia de las clases, se efectúan una serie de comprobaciones diferentes y el **gestor de seguridad** de la VM prohíbe los accesos peligrosos.

  - Nivel de seguridad *dado por la API de Java*. El conjunto de métodos y clases que estamos obligados a utilizar cuando programamos en Java para acceder a los recursos del sistema, está definido por la API, y constituye la última barrera defensiva. El diseño de dichos métodos y clases hace que éstos realicen múltiples verificaciones cuando son invocados, de modo que se dificultan los errores (voluntarios o involuntarios).

- **Adaptación a redes (y en particular a Internet).** Los applets (componentes de software que corren en el contexto de otro programa) disponen de una significativa riqueza de recursos y son capaces de realizar **tareas muy complejas** a pesar de su reducido tamaño. Una de las explicaciones de esta sorprendente capacidad es el hecho de que los applets se sirven del propio código del navegador en cuya VM se ejecutan, utilizándolo para tareas tales como presentación gráfica o comunicaciones. Sin embargo, el acceso a las funciones del navegador es totalmente automático y transparente para el programador, que debe limitarse a invocar ciertas funciones de la API de Java; estas invocaciones, interpretadas por el navegador, dan origen a acciones muy complejas. Esta observación es muy importante cuando se discute del **rendimiento** de Java, pues todas estas acciones se realizan en la máquina que está ejecutando el applet, y la rapidez de ejecución de las mismas no depende de que Java sea un lenguaje semiinterpretado (o semicompilado).

3.1.2 Herramientas para gestionar la base de datos

Herramientas utilizadas para la administración y el uso de la base de datos:

- Servidor de bases de datos MySql 5.0
- Gestor gráfico SQLYog 5.02
- Conector JDBC
- Api JDBC
- Pull de conexiones

MYSQL 5.0

Como servidor de bases de datos, el equipo de desarrollo decidió emplear MySQL 5.0. La razón fundamental para elegirlo fue que todos habíamos trabajado en anteriores proyectos tanto con MySQL como con el gestor gráfico SQLYog, que comentaremos a continuación.

Existen otros dos motivos clave por los cuales se decidió utilizar esta herramienta: se trata de un software libre y robusto. MySQL es un software de código abierto que cuenta con licencia GPL (General Public License) de que dispone y la calidad de este servidor de bases de datos. Se trata de uno de los servidores más robustos, fiables y rápidos que existen en la actualidad, tanto para volúmenes de datos grandes como pequeños. Su integración con el lenguaje Java resulta muy sencilla.

Se trata de un Sistema de Gestión de Bases de Datos relacional, modelo que se caracteriza muy a grandes rasgos por la estructuración a nivel lógico de los datos en tablas, formadas por filas y columnas, aunque a nivel físico pueden tener una estructura totalmente diferente.

BREVE HISTORIA DE MYSQL

En 1981 IBM empezó a comercializar el lenguaje SQL, imprescindible para el desarrollo de las bases de datos relacionales. Dos años después, en 1983, nació DB2, la base de datos relacional más popular.

No fue hasta 1990 cuando surgiría MySQL: el programador sueco Michael Windenis comenzó a usar mSQL para conectar tablas usando sus propias rutinas de bajo nivel (ISAM). Pero esto no era suficientemente rápido ni potente y decidió crear su propio gestor de datos para la aplicación que estaba desarrollando.

En poco tiempo, Mysql se convertiría en el número 1 en el ámbito de las bases de datos de código libre. Al cabo de los años se creó una mepresa sueca llamada Tux, que comercializó nuevas versiones de MySQL.

PRINCIPALES CARACTERÍSTICAS DE MYSQL

A continuación se enumeran brevemente algunas de las características más destacables de MySQL:

- Escrito en C y C++, testado con GCC 2.7.2.1. Usa GNU autoconf para potabilidad.
- Clientes C, C++, JAVA, Perl, TCL.
- Aprovecha la potencia de sistemas multiprocesador, gracias a su implementación multihilo.
- Soporta gran cantidad de tipos de datos para las columnas.
- Dispone de API's en gran cantidad de lenguajes (C, C++, Java, PHP, etc).
- Gran portabilidad entre sistemas.
- Soporta hasta 32 índices por tabla.
- Gestión de usuarios y passwords, manteniendo un muy buen nivel de seguridad en los datos.
- Puede trabajar en distintas plataformas y S.O. distintos.
- Sistema de contraseñas y privilegios muy flexible y seguro.
- Registros de longitud fija y variable.
- Todas las columnas pueden tener valores por defecto.
- Utilidad (Isamchk) para chequear, optimizar y reparar tablas.
- Todos los datos están grabados en formato ISO8859_1.
- Los clientes usan TCP o UNIX Socket para conectarse al servidor.
- El servidor soporta mensajes de error en distintas lenguas.
- Todos los comandos tienen -help o -? para las ayudas.
- Diversos tipos de datos: enteros, coma flotante, doble precisión, carácter, fechas, enumerados, etc.

La última versión disponible de MySQL a Mayo de 2006 en http://www.mysql.com/ es la 5.0, aunque también se puede descargar la versión alfa de MySQL 5.1. En este proyecto se ha trabajado con la anterior.

Algunas de las novedades de la versión 5.0 utilizada en este proyecto son:

- Procedimientos almacenados y funciones del SQL
- Triggers
- Vistas
- Cursors
- Esquemas de información
- XA Transacciones distribuidas
- modo SQL
- Nuevos motores para almacenamientos de archivos.
- Nuevas herramientas para migración.
- Administrador de Instancias.
- Conectores y herramientas visuales actualizadas.
- Tipos de datos ARCHIVE y FEDERATED

SQLYOG 5.X

SqlYog es una herramienta gráfica diseñada para administrar de forma fácil y rápida el servidor de bases de datos MySQL. La razón por la que elegimos esta herramienta, como ya se ha comentado anteriormente, fue el conocimiento de la misma por parte de todos los miembros del equipo de desarrollo.

Actualmente SQLyog es un producto comercial, sin embargo, es posible conseguir una versión freeware. Se utilizará la versión 5.01, disponible en http://www.webyog.com/.

Principales características de SQLYog

1. Ver y explorar las tablas de una base de datos.
2. Crear y editar índices.
3. Exportar los resultados de una consulta a un archivo.
4. Coloreado de sintaxis SQL.
5. Crear y eliminar bases de datos.
6. Crear, eliminar y renombrar tablas.
7. Insertar y editar registros en una tabla de manera gráfica.
8. Crear usuarios y editar los privilegios de los ya existentes.

EL DRIVER JDBC

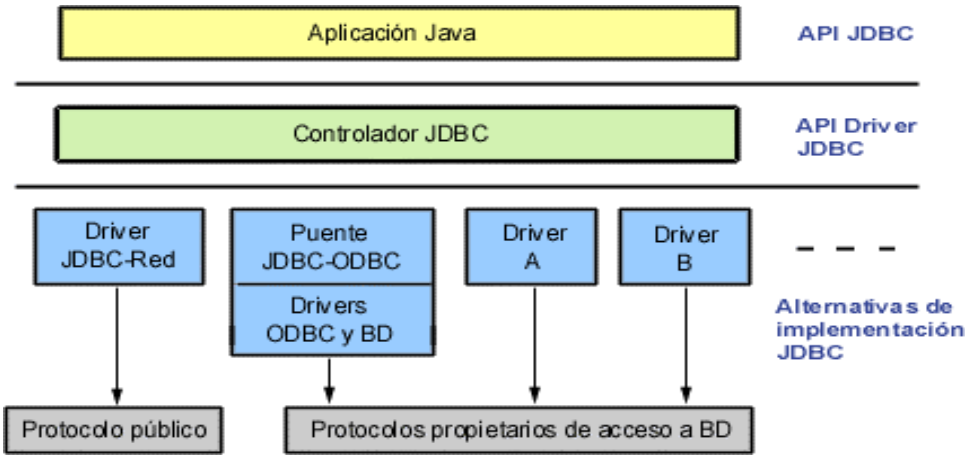
Los distribuidores de bases de datos suministran los controladores que implementan el API JDBC y que permiten acceder a sus propias implementaciones de bases de datos. En el caso del manejador de bases de datos MySQL, Connector/J es el controlador JDBC oficial.

MySQL Connector/J es un driver nativo de Java que convierte las llamadas JDBC al protocolo de red usado por el manejador de bases de datos MySQL. Este driver permite que los desarrolladores que trabajan con Java construyan programas y applets que interactúen con MySQL y les permita conectar todos los datos coorporativos, aún en un ambiente heterogéneo. MySQL Connector/J es un driver JDBC del tipo IV y tiene un completo conjunto de características que soporta sin inconvenientes las capacidades de MySQL.

EL API JDBC

JDBC son las siglas de Java Database Connnectivity, un API de Java que permite el acceso a bases de datos y a casi cualquier tipo de dato tabular desde el lenguaje de programación Java. El paquete java.sql, que contiene las clases e interfaces que permiten acceder a la funcionalidad básica de JDBC. Forma parte de la edición estándar de J2SE.

Este API consiste en un conjunto de clases e interfaces Java que permiten a un programa Java acceder a diferentes tipos de bases de datos de forma homogénea. La aplicación escrita en Java debe poder acceder a un controlador o driver, que será el encargado de implementar la funcionalidad de las clases de acceso a datos y que será el encargado de gestionar la comunicación real entre el API y la base de datos.



Veamos de forma resumida cuales son los pasos para realizar una conexión en Java con una base de datos cualquiera:

1. Cargar el driver JDBC: Para conectarse a una base de datos a través de JDBC desde un programa Java, lo primero que se necesita es cargar el driver, que se encargará de convertir la información que se envía a través de la aplicación a un formato que lo entienda la base de datos. Esta parte del código sería la única que dependería del tipo de driver y del tipo de base de datos:

```
Class.forName("Clase  del driver").newInstace();
Class.forName("org.gjt.mm.mysql.Driver");
```

2. Conectarse a la Base de Datos utilizando la clase Connection: Para conectarse a una fuente de datos específica, una vez cargado el driver, se utiliza una URL:

```
Connection con = DriverManager.getConnection(URL, usuario, password);
```

3. Crear sentencias SQL, utilizando objetos de tipo Statement: Para recuperar información de una de base de datos se utiliza la clase Statement, cuyos objetos se crean a partir de una conexión y tienen el método executeQuery para ejecutar consultas SQL de tipo SELECT devolviendo como resultado un conjunto de registros en un objeto de la clase ResultSet.

```
s = con.createStatement();
rs = s.executeQuery("SELECT * FROM GenomeProyect ORDER BY nombre");
```

POOL DE CONEXIONES

En las aplicaciones en las que es necesario el acceso concurrente a una base de datos es necesario disponer de varias conexiones. El proceso de creación y destrucción de una conexión a una base de datos es costoso e influye sensiblemente en el rendimiento de una aplicación. La estrategia más utilizada en este tipo de aplicaciones es el uso de un pool de conexiones.

Básicamente, un pool de conexiones es una clase JAVA que se basa en un vector de conexiones, controlando el número de conexiones libres y el número de conexiones ocupadas, haciendo balanceo de dichas conexiones. Cuando un cliente solicita una conexión, comprueba si hay una conexión disponible. En ese caso reserva la conexión. Una vez utilizada la conexión, la libera, dejándola disponible a otros usuarios.

Normalmente, existen dos métodos para controlar un pool de conexiones:

- getConnection(): este método retorna una conexión (si esta disponible).
- freeConnection(): este método libera la conexión que estamos usando.

3.1.3 El servidor: TOMCAT 4.1.31

Tomcat (también llamado *Jakarta Tomcat* o *Apache Tomcat*) funciona como contenedor de servlets y es desarrollado bajo el proyecto Jakarta en la *Apache Software Foundation*. Implementa las tecnologías *Java Servlet* 2.3 y *JavaServer Pages 1.2* (JSP) de *Sun Microsystems*.

Tomcat es un servidor de aplicaciones que, a diferencia de un servidor Web, como es por ejemplo *Apache*, incluye un contenedor Web que puede servir páginas dinámicas (a diferencia del servidor Web, que solo sirve páginas HTML estáticas). Incluye el compilador Jasper, que compila páginas JSPs y las convierte en servlets. Además, funciona con cualquier sistema operativo que disponga de máquina virtual Java, ya que fue escrito en este mismo lenguaje.

HISTORIA

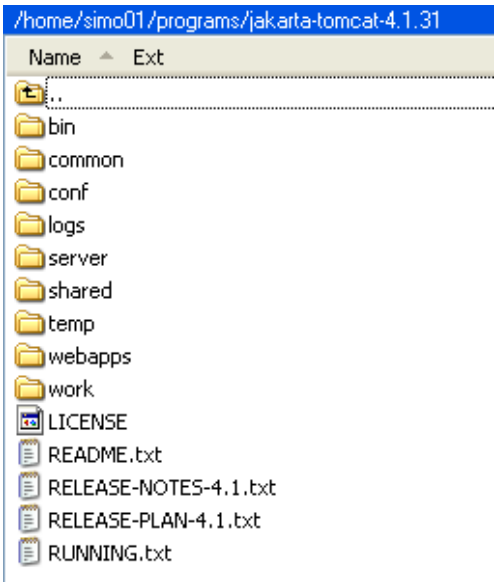
James Duncan Davidson se puede considerar el padre de Tomcat. Trabajaba como arquitecto de software para Sun Microsystems en el momento en que decidió donar el proyecto a la Apache Software Foundation.

Eligió el nombre de Tomcat (gato) pretendiendo representar la capacidad del programa de ser independiente, de cuidarse por sí mismo. Aunque el verdadero motivo es que Duncan esperaba que Tomcat se convirtiese en *open source* y que O`Reilly (famosa editorial norteamericana dedicada a la informática) publicase un libro sobre su proyecto. O`Reilly es conocida por asociar animales a las portadas de sus libros.

USO

Al directorio "C:\Proyecto\Jakarta Tomcat 4.1.31", que es el directorio base de Tomcat se le denomina CATALINA_HOME, y para referirse a él se escribe $CATALINA_HOME.

Si revisamos el contenido del directorio de instalación de Tomcat veremos una estructura de directorios como la que se muestra en la siguiente figura. En ésta encuentra el directorio **webapps** sobre el que trabajaremos posterioremente.



A continuación se describe brevemente cada uno de estos directorios.

Directorio	Descripción
Bin/	Incluye los archivos binarios y scripts de Tomcat.
common/	Para clases disponibles para Catalina y aplicaciones web.
conf/	Guarda los archivos de configuración.
Logs/	Guarda los archivos de registros (logs).
server/	Contiene clases para uso interno y exclusivo de Catalina.
shared/	Contiene clases compartidas por todas las aplicaciones web.
temp/	Guarda archivos temporales creados por la Máquina Virtual de Java.
webapps/	Directorio base para las aplicaciones web.
work/	Directorio de trabajo para archivos y directorios temporales de Tomcat.

El nombre de **Catalina** se refiere al contenedor de servlets en sí que se incluye con Tomcat.

En algunos de estos directorios se encuentran los subdirectorios **classes** y **lib**. En el primero se colocan las clases "sueltas", sin empaquetar, y en el segundo se colocan clases empaquetadas en archivo JAR.

Una vez lanzado Tomcat, si no hubo ningún problema, debe estarse ejecutando en nuestra máquina el servidor web que se incluye con Tomcat, y que como se explicó anteriormente, es el que nos permitirá ejecutar los servlets que escribiremos más adelante.

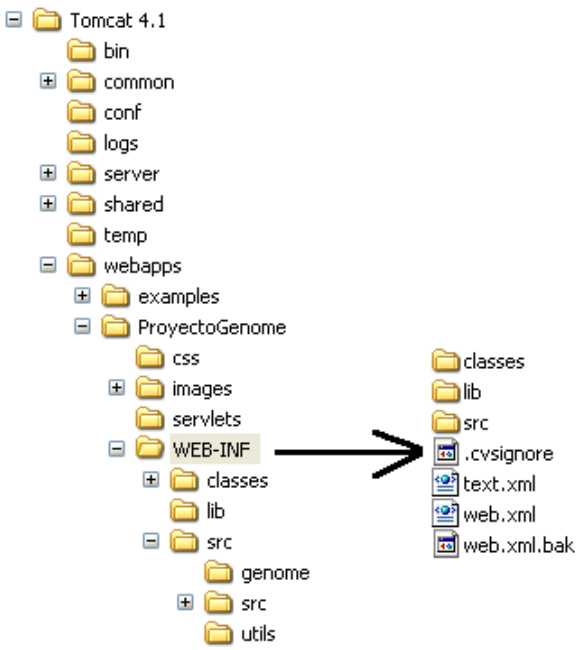
Un servidor web escucha peticiones en un puerto concreto, normalmente el puerto 80, aunque puede usar prácticamente cualquier otro número de puerto. Si un servidor web está escuchando en otro puerto distinto del 80, las peticiones dirigidas al servidor deben especificar el número de puerto en concreto.

Según la instalación base, Tomcat escuchará peticiones en el puerto 8080. De modo que desde el navegador podemos acceder a Tomcat con el siguiente URL:

http://localhost:8080/

Para que una aplicación web pueda ser ejecutada, ésta debe ser desplegada en el contenedor de servlets. Existen varias maneras en las que una aplicación web puede ser desplegada en Tomcat, nosotros hablaremos únicamente de una de ellas, la que se basa en la creación de una estructura de directorios preestablecida bajo el directorio **webapps**, ya que es la más sencilla y rápida.

En este caso, Tomcat le asigna una ruta al contexto de la aplicación, basada en el nombre del subdirectorio que se elija. La estructura de directorios está basada en la especificación del API Java Servlet, y cualquier contenedor de servlets debería de seguir estas reglas. El nivel superior, o directorio raíz de la aplicación contendrá documentos HTML, scripts JSP, imágenes, y algunos otros archivos. A partir de éste, se pueden tener todos los subdirectorios que se requieran. Observar la siguiente figura.



El directorio raíz contiene un directorio especial llamado **WEB-INF**. Este directorio no es visible para los usuarios de la aplicación, sin embargo, aquí se guardan todas las clases, servlets y archivos JAR de los que conste una aplicación web (por ejemplo "text.xml, donde guardamos el texto de la web en los dos idiomas). Dentro del directorio WEB-INF hay dos subdirectorios y un archivo que son de especial interés:

- **classes** - Este directorio contiene servlets y otras clases. Estas clases se hayan automáticamente por el cargador de servlets como si estuvieran en la ruta de clases (CLASSPATH). Puede incluir subdirectorios que correspondan a la estructura de paquetes.

- **lib** - Es similar al directorio anterior, pero contiene únicamente archivos JAR.

- **web.xml** - Es un documento XML llamado *descriptor de despliegue*. Tiene una estructura rigurosamente definida que se usa para configurar los servlets y otros recursos que forman parte de una aplicación web.

De acuerdo con esta estructura de directorios, el archivo connector.jar que contiene el driver JDBC para MySQL se colocará en el directorio ProyectoGenome/WEB-INF/lib. De hecho, todos los archivos JAR que vayan a ser usados por nuestra aplicación web deben colocarse en este mismo directorio. Todos los servlets y otras clases de nuestra aplicación que no estén en archivos JAR deben ser colocados en el directorio ProyectoGenome/WEB-INF/classes.

Cualquier otro archivo (HTML, GIF, JPG, etc) se debe colocar en el directorio ProyectoGenome.

Es muy importante señalar que por defecto Tomcat no vuelve a cargar los servlets si han sufrido alguna modificación y se han vuelto a compilar. Por ejemplo, si modificamos el programita HolaMundo.java y lo compilamos nuevamente, al invocarlo desde el navegador no veremos ningún cambio. Dado que esto es sumamente importante y necesario cuando se están desarrollando y depurando los servlets, a continuación mostraremos como hacer para que Tomcat vuelva a cargar los servlets que hayan sufrido algún cambio. Para ello, tenemos que editar el archivo server.xml que está en el directorio $CATALINA_HOME/conf, y agregamos la siguiente directiva apróximadamente en la línea 268, entre el contexto ROOT y el contexto Examples.

```
<DefaultContext reloadable="true"/>
```

De esta manera, para cada aplicación web que se cree de la misma manera que la nuestra, se tendrá habilitada la recarga de servlets. Cabe mencionar que por razones de eficiencia, esta característica de Tomcat debería de usarse únicamente para sistemas que no se encuentran en producción, es decir, que se encuentran en la etapa de desarrollo.

3.1.4. WEB SERVICES (SOAP)

¿QUÉ SON LOS WEB SERVICES?

Son muchas las definiciones que se pueden hacer de un web services. Algunas de ellas son:

- Según la **W3C**: “Un servicio Web es una aplicación software identificada mediante una URI, cuyo interfaz (y uso) es capaz de ser definido, descrito y descubierto mediante artefactos XML, y soportar interacciones directas con otras aplicaciones software usando mensajes basados en XML y protocolos basados en Internet”.

- Otra definición sería:

  Un servicio Web es un componente software que se basa en las siguientes tecnologías:

  - Un formato que describa la interfaz del componente (sus métodos y atributos) basado en XML. Por lo general este formato es el WSDL (Web Service Description Language).
  - Un protocolo de aplicación basado en mensajes y que permite que una aplicación interaccione (use, instancia, llame y ejecute) al web service. Por lo general este protocolo es SOAP (Simple Object Access Protocol).
  - Un protocolo de transporte que se encargue de transportar los mensajes por internet. Por lo general este protocolo de transporte es HTTP (Hiper-Text Transport Protocol) que es exactamente el mismo que se usa para navegar por la Web.

Pero más claramente tenemos esta otra definición:

- Los Web Services son pequeños programas formados por varios componentes que permiten ser publicados en directorios e invocados para su ejecución por otros programas vía http, generando una respuesta en XML.

¿ POR QUÉ WEB SERVICES?

En los últimos años, los sistemas que soportan las aplicaciones de negocio de las empresas han visto cómo crecían de forma exponencial las relaciones de la empresa con su entorno - clientes y proveedores - forzando su adaptación a este nuevo marco de relación.

En su origen, los Web Services fueron creados como un método para compartir recursos en la red. En un entorno donde el aumento constante del número de usuarios demandaba cada vez más un mayor número de recursos en la red, surgió la necesidad de facilitar la distribución entre empresas de dichos recursos para satisfacer las necesidades de sus clientes. El resultado fue el desarrollo de una tecnología de muy fácil implantación y que era capaz de solucionar los aspectos de disponibilidad e inmediatez que se requerían.

Esta tecnología ha tenido una aceptación bastante importante excepto para los servicios que implicaban transacciones seguras, debido a que aún se están definiendo los estándares para asegurar el acceso a los Web Services. Tal es así, que este planteamiento se está empezando a trasladar a la Intranet de las empresas. Así, los Web Services se están revelando como la tecnología capaz de distribuir los recursos internos entre todos los sistemas, ahorrando costosos desarrollos de integración.

En conclusión, los web services aportan grandes ventajas porque permiten que varias aplicaciones:

1. Compartan información
2. Invoquen funciones de otras aplicaciones independientemente de:

    - Cómo hayan sido creadas (lenguaje de programación)
    - Cómo se ejecutan (sistema operativo y plataforma)
    - Dispositivos utilizados para acceder a ellas

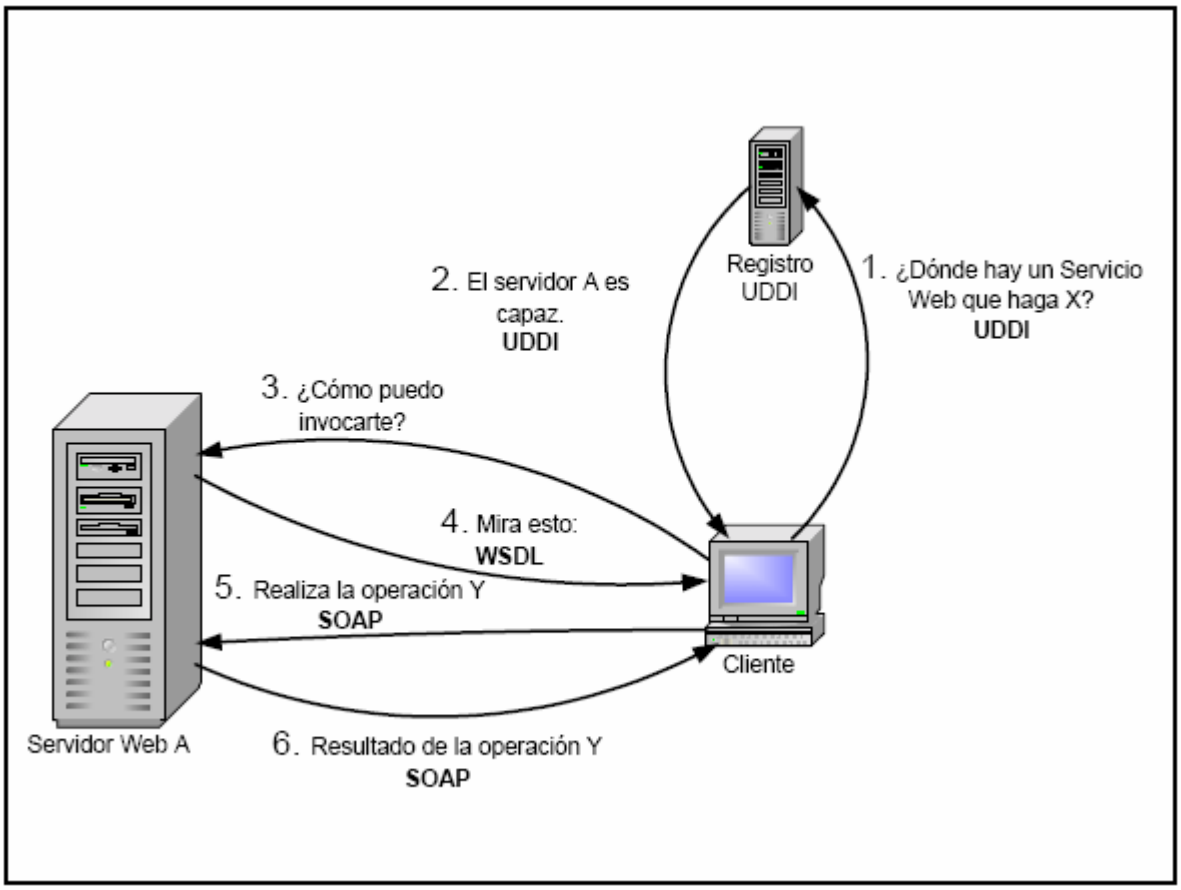
LAS 4 CAPAS DE UN WEB SERVICE.

Como hemos indicado anteriormente en un Web Service intervienen otras tecnologías que van a definir las 4 capas en las que se basa. Estas 4 capas son:

- **Descubrimiento**: encontrar un Web Service que haga lo que quiero (UDDI).
- **Descripción**: pedirle al Web Service que se describa (para saber cómo es la invocación, WSDL).
- **Invocación**: invocar el método (SOAP).

- **Transporte**: transporte de la invocación y la respuesta. Generalmente HTTP.

En el siguiente dibujo vemos los sucesivos pasos:



Dibujo 1. Las 4 capas de un Web Service.

¿QUÉ SON SOAP, WSDL Y UDDI?

Estos elementos intervienen en el proceso de un web service y vamos a explicarlos más detalladamente a continuación.

SOAP

Es el acrónimo de Simple Object Access Protocol, es decir protocolo simple de acceso a objetos. SOAP es un protocolo ligero de mensajes XML que se usa para codificar la información de los mensajes de petición y respuesta de los Web Services que se envían a través de una red.

Los mensajes SOAP son independientes de los sistemas operativos y de los protocolos, y pueden ser transportados usando una variedad de protocolos internet, incluyendo SMTP, y HTTP.

Dentro del paradigma orientado a objetos, usar un Web Service es igual que usar cualquier otra clase. Y esto significa instanciarlo, y llamar a sus métodos, pasándoles los parámetros que sean necesarios, y obteniendo a su vez el resultado que nos devuelvan.

Generalmente llamaremos a Web Services que no están en nuestra máquina local, sino en cualquier servidor accesible desde internet. Debemos, por tanto, de disponer de alguna forma de llamar a cualquiera de sus métodos pasándole los parámetros oportunos y obteniendo el resultado de esa llamada (si es que el método devuelve algo después de ser ejecutado).

Soap es un protocolo que define precisamente cómo realizar esta comunicación, es decir cómo debemos codificar las llamadas a los métodos de un web service, y cómo debe el web service codificar el resultado para que nosotros lo podamos interpretar. Estos mensajes son los que transportarán los protocolos de transporte, por lo general HTTP.

SOAP ha recibido gran atención debido a que facilita una comunicación del estilo RPC entre un cliente y un servidor remoto. Pero existen multitud de protocolos creados para facilitar la comunicación entre aplicaciones, incluyendo RPC de Sum, DCE de Microsoft, RMI de Java y ORPC de CORBA. ¿Por qué se presta tanta atención a SOAP?

Una de las razones principales es que SOAP ha recibido un gran apoyo por parte de la industria. SOAP es el primer protocolo de su tipo que ha sido aceptado prácticamente por todas las grandes compañías de software del mundo. Compañías que en raras ocasiones cooperan entre sí están ofreciendo su apoyo a este protocolo. Algunas de las mayores Compañías que soportan SOAP son Microsoft, IBM, Sun Microsystems, SAP y Ariba.

Algunas de las ventajas de SOAP son:

- **No esta asociado con ningún lenguaje:** El lenguaje de programación es independiente. SOAP no especifica una API, por lo que la implementación de la API se deja al lenguaje de programación, como en Java, y la plataforma como Microsoft .Net.

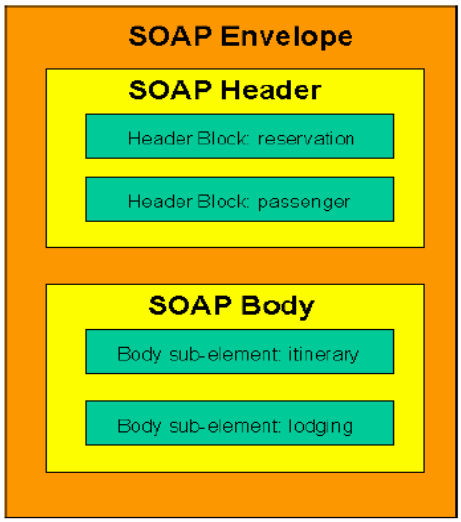
- **No se encuentra fuertemente asociado a ningún protocolo de transporte:** La especificación de SOAP no describe como se deberían asociar los mensajes de SOAP con HTTP. Un mensaje de SOAP no es más que un documento XML, por lo que puede transportarse utilizando cualquier protocolo capaz de transmitir texto.

- **Aprovecha los estándares existentes en la industria:** Por ejemplo, SOAP aprovecha XML para la codificación de los mensajes, en lugar de utilizar su propio sistema de tipo que ya está definido en la especificación esquema de XML. Los mensajes de SOAP se pueden asociar a los protocolos de transporte existentes como HTTP y SMTP.

- **Permite la interoperabilidad entre múltiples entornos:** SOAP se desarrollo sobre los estándares existentes de la industria, por lo que las aplicaciones que se ejecuten en plataformas con dicho estándares pueden comunicarse mediante mensaje SOAP con aplicaciones que se ejecuten en otras plataformas.

¿Cómo es un mensaje SOAP?

La estructura general de un mensaje SOAP tiene como elemento raíz del documento el elemento **Envelope**. La raíz contiene a su vez dos subelementos, **Header** y **Body**.



El "Envelope" contiene un elemento *Header* (opcional) que guarda información sobre el mensaje. También contiene un elemento (éste es obligatorio) llamado *body*. Dicho elemento contiene la carga de datos del mensaje.

Los usos típicos de la parte body son proveer un mecanismo simple de intercambiar información con el receptor del mensaje SOAP. En esta parte del mensaje es donde se encuentran las invocaciones RPC o bien el resultado de la invocación.

Un ejemplo de la estructura de mensaje SOAP es el siguiente:

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
    SOAP-ENV: encodingStyle= http://schemas.xmlsoap.org/soap/encoding/ >
    <SOAP-ENV:Header>
            <User Information>
    </SOAP-ENV:Header>
   <SOAP-ENV:Body>
          <m1:RemoteMethodName   xmlns:m1="http://www.sitioweb.com#">
               <arg1>value1</arg1>
               <arg2>value2</arg2>
          </m1:RemoteMethodName>
        <m2:RemoteMethodName  xmlns:m2="http://www.sitioweb.com#">
               <arg1>value1</arg1>
         </m2:RemoteMethodName>
    </SOAP-ENV:Body>
</SOAP:Envelope>
```

WSDL

Es un acrónimo de Lenguaje de Descripción de Servicios Web (Web Services Description Language), que es un lenguaje XML usado para describir la interfaz de un web service como un conjunto de puntos finales de comunicación (métodos) capaces de intercambiar mensajes (es decir recibir llamadas con sus parámetros correspondientes y generar respuesta con el resultado que le corresponda).

WSDL es el lenguaje usado por UDDI para describir a los web services. Fue desarrollado de forma conjunta por Microsoft e IBM.

WSDL es un fichero XML que describe el conjunto de métodos expuestos por un web service. Esta descripción incluye el número de argumentos, y tipo de cada uno de los parámetros de cada uno de los métodos, así como la descripción de los elementos que devuelve. Estas descripciones son las que se usan para generar los objetos proxy que se usan en los entornos de desarrollo con los que se programan web services.

Por cada web service, cogemos su descripción WSDL y generamos una clase con la misma interfaz (igual número de métodos y la misma signatura de los mismos) que describe el fichero. Esta clase es el proxy local del web service. El código local de un proxy de web service es el encargado de construir las llamadas SOAP al servicio web y de recepcionar las llamadas SOAP de ese servicio Web.

Usando este patrón es posible abstraerse de todos los elementos que intervienen en una llamada a un servicio Web, ya que utilizar este patrón es exactamente igual a llamar a una clase local (el proxy) y es éste el que se encarga de encapsular la complejidad propia de la comunicación con el servicio Web.

Las clases proxys son generadas por lo general de forma automatica por la mayoría de los entornos de desarrollo.

Un documento WSDL contiene 4 partes principales que indican **qué** hace el servicio, qué **tipo** de datos utiliza en sus mensajes, **cómo** se comunica y **donde** reside:

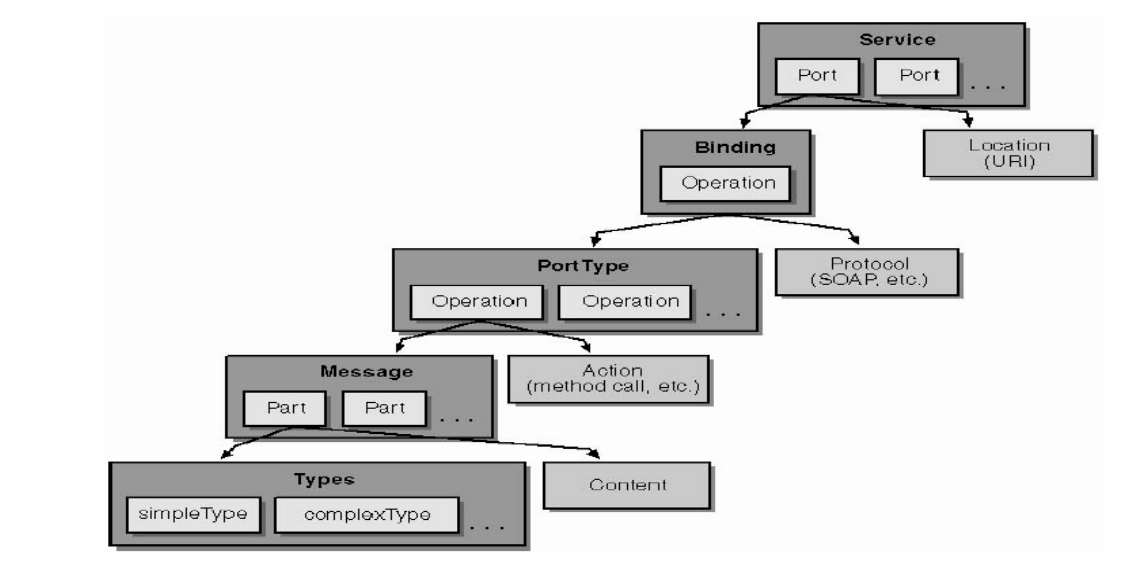
- **Qué hace el servicio**: Proporciona información sobre la interfaz exportada. Define el mensaje. Consta de elementos **message**, y **portType**.
- **Tipo de datos que utiliza el mensaje**: Define los tipos de datos soportados por los mensajes intercambiados (peticiones, respuestas, errores, etc.). Consta de elementos **types.**
- **Cómo se comunica**: Describe los detalles de la implementación técnica de nuestro Web Service. Indica cómo transportar los mensajes en la comunicación. Consta de elementos **binding**. Un binding une un portType a un protocolo de comunicación (Ej. SOAP sobre HTTP).

- **Dónde reside**: Indica la localización del servicio. Consta del elemento **service**, y coloca juntos el **portType**, el **binding**, y la **localización** real (una URI) del Web Service.

Un documento WSDL contiene las siguientes definiciones:

- **Definitions:** elemento raíz del documento WSDL. Se usa para declarar los espacios de nombres.
- **Types:** se usa para describir tipos de datos para los mensajes intercambiados.
- **Message:** descripción de los datos que van a ser transmitidos. Son una colección de "data values" de un tipo particular (utilizando XML Schema como mecanismo de tipación).
- **Operation:** descripción abstracta de la acción soportada por el servicio.
- **PortType:** Colección de "operations" o "signatures" de los métodos que definen el intercambio ordenado de los mensajes.
- **Bindings:** especifica los protocolos que usa cada "port" y el "encoding".
- **Port:** "PortType" + "Binding". Es una descripción de una acción soportada por el servicio. Cada operación se corresponde a un mensaje de input o de output.
- **Service:** Conjunto de "ports" relacionados que implementan el servicio.

A continuación se muestra un gráfico representativo de la estructura de un documento WSDL:



(Dibujo2: componentes de un documento WSDL).

UDDI

Es un acrónimo de Integración, Descubrimiento y Descripción Universal (Universal Description, Discovery and Integration). Es un directorio de Web Services distribuido y basado en Web que permite que se listen, busquen y descubran este tipo de software.
UDDI tiene 2 partes:

- Un directorio con los metadatos de todos los Web Services, incluyendo un puntero a la descripción WSDL de cada uno.
- Las definiciones de "PortTypes" WSDL para manipular y buscar en ese directorio.

Define estándares para un registro distribuido de servicios Web. Puede verse como la suma de:

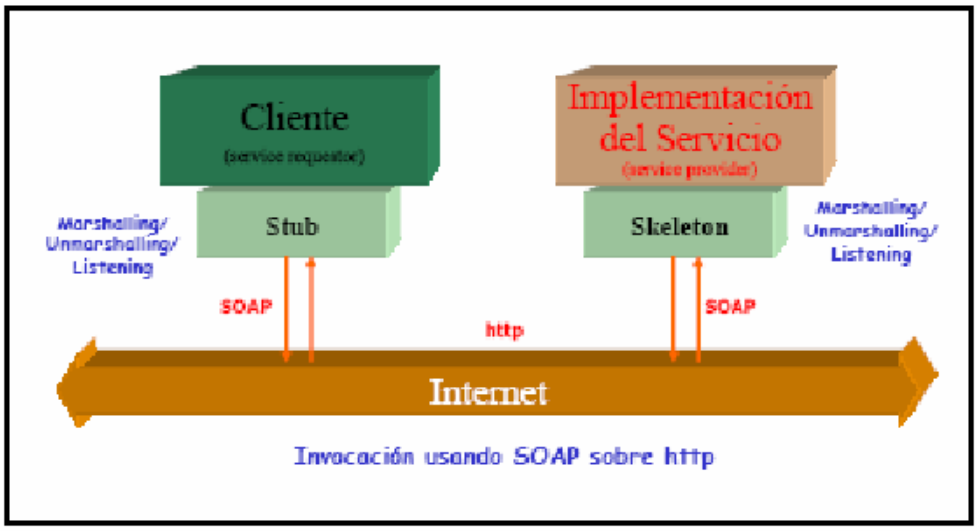
- **White pages** (información general): Contiene dirección e información de contacto del proveedor del servicio.
- **Yellow pages** (categorías de servicios): Permiten buscar servicios por categorías y clasificaciones
- **Green pages** (reglas de negocio): Contienen suficiente información técnica sobre los servicios ofrecidos para permitir su invocación

¿Cómo funciona la invocación?

Cuando se quiere invocar a un web service y ya se sabe su estructura (bien mediante WSDL o bien porque el web service lo haya creado uno mismo), lanzamos un mensaje SOAP con los parámetros adecuados, este viaja a través de HTTP y llega hasta el servidor que contiene el web service requerido. Dicho web service ejecuta la función descrita en el mensaje

SOAP y devuelve los datos resultantes al Cliente mediante otro mensaje SOAP y enviado a través de HTTP.

Dibujo 2. Invocación de un Web Service.



3.2. Tecnologías y Herramientas WEB

3.2.1. HTML (HyperText Markup Language)

INTRODUCCION A HTML

Para poder empezar a explicar, que es JSP, JAVASCRIPT, CSS y otras tecnologías empleadas en nuestro proyecto, debemos introducirnos un poco en el mundo World Wide Web (www) y la tecnología básica HTML que entrelaza y emplea a todas las demás.

¿Qué es la World Wide Web?

La *World Wide Web (Web)* es una red de recursos de información. La Web se basa en tres mecanismos para hacer que estos recursos estén listos y a disposición de la mayor audiencia posible:

1. Un esquema uniforme de nombres para localizar recursos en la Web (por ejemplo URIs, ó, más en concreto, las URLs).
2. Protocolos, para acceder a recursos con nombre en la Web (p.ej., HTTP).
3. Hipertexto, para navegar fácilmente entre recursos (p.ej., HTML).

¿Qué es el HTML?

HTML es el lenguaje con el que se definen las páginas web. Básicamente se trata de un conjunto de etiquetas que sirven para definir la forma en la que presentar el texto y otros elementos de la página.

El HTML se creó en un principio con objetivos divulgativos. No se pensó que la web llegara a ser un área de ocio con carácter multimedia, de modo que, el HTML se creó sin dar respuesta a todos los posibles usos que se le iba a dar y a todos los colectivos de gente que lo utilizarían en un futuro. Sin embargo, pese a esta deficiente planificación, se han ido incorporando modificaciones con el tiempo, esto es a lo que llamamos estándares del HTML.

En definitiva, para publicar información y distribuirla globalmente, se necesita un lenguaje entendido universalmente, una especie de lengua franca de publicación que todas las computadoras puedan comprender potencialmente. El lenguaje de publicación usado por la World Wide Web es el HTML.

El HTML da a los autores las herramientas para:

- Publicar documentos en línea con encabezados, textos, tablas, fotos, etc.
- Obtener información en línea a través de vínculos de hipertexto.

- Diseñar formularios para realizar transacciones con servicios remotos, para buscar información, hacer reservas, pedir productos, etc.
- Incluir hojas de cálculo, videoclips, sonidos, y otras aplicaciones directamente en sus documentos.

Breve historia del HTML

El HTML fue desarrollado originalmente por *Tim Berners-Lee* mientras estaba en el CERN, y fue popularizado por el navegador Mosaic desarrollado en el NCSA. Durante los años 90 ha proliferado con el crecimiento explosivo de la Web.

La Web depende de que los autores de páginas Web y las compañías compartan las mismas convenciones de HTML. Esto ha motivado el trabajo colectivo en las especificaciones del HTML.

El HTML 2.0 (noviembre de 1995) fue desarrollado bajo los auspicios de la Internet Engineering Task Force (IETF) para codificar lo que era la práctica común a finales de 1994. HTML+ (1993) y HTML 3.0 (1995, ver [HTML30]) propusieron versiones mucho más ricas de HTML. A pesar de no haber logrado nunca el consenso en las discusiones sobre estándares, estos borradores llevaron a la adopción de buen número de nuevas características. Los esfuerzos del Grupo de Trabajo HTML del World Wide Web Consortium para codificar la práctica común en 1996 condujeron a HTML 3.2 (enero de 1997, ver [HTML32]).

La mayoría de las personas están de acuerdo en que los documentos HTML deberían funcionar bien en diferentes navegadores y plataformas. Gracias a la interoperabilidad los proveedores de contenidos reducen gastos, ya que sólo deben desarrollar una versión de cada documento. Si este esfuerzo no se realiza, hay un riesgo mucho mayor de que la Web se convierta en un mundo propietario de formatos incompatibles, que al final acabaría por reducir el potencial comercial de la Web para todos los que forman parte de ella.

El HTML ha sido desarrollado con la premisa de que cualquier tipo de dispositivo debería ser capaz de usar la información de la Web: PCs con pantallas gráficas con distintas resoluciones y colores, teléfonos móviles, dispositivos de mano, dispositivos de salida y entrada por voz, computadoras con anchos de banda grandes o pequeños, etc.

El actual HTML 4.

El HTML 4 desarrolla el lenguaje HTML con mecanismos para hojas de estilo, ejecución de scripts, marcos, objetos incluidos, soporte mejorado para texto de derecha a izquierda y direcciones mezcladas, tablas más ricas y mejoras en formularios, ofreciendo mejoras de accesibilidad para personas con discapacidades.

SINTAXIS HTML

Al igual que diremos en otros apartados posteriores, no vamos a entrar a definir detalladamente la sintaxis del lenguaje HTML por no ser el objeto de este trabajo. Sólo entraremos un poco en detalle en su estructura básica y a dos elementos en particular: los formularios y los frames (marcos).

Las directivas en HTML son aquellas "ordenes" que se hayan entre los símbolos <...> y que son de 2 clases: abiertas y cerradas.

- **Abiertas** (ó vacías según bibliografía)

  Son aquellas directivas que no necesitan ser cerradas para que su acción sea comprendida por el navegador (<HR>,
, ...)

- **Cerradas** (o contenedoras)

  Son aquellas directivas que para ser comprendidas por el navegador, deben indicar dónde comienzan y terminan. El texto o imágenes o elementos que se encuentren entre las 2 directivas <DIRECTIVA>...</DIRECTIVA> quedan afectadas por la acción de dicha DIRECTIVA en su totalidad (<H1>...</H1>, <TABLE>...</TABLE>, ...).

Todo el contenido de un documento HTML debe encontrarse englobado en el interior de la directiva cerrada <HTML>...</HTML>. Esto indica al navegador que estamos empleando que todo lo que se encuentra entre estas dos marcas, es código HTML y que debe ser interpretado como tal. La directiva <HTML>...</HTML> contiene en su interior 2 bloques: <HEAD>...</HEAD> que es la directiva que señala la cabecera del documento y, <BODY>...</BODY>, que contiene el cuerpo del documento o para entendernos lo que visualizamos en el navegador con el formato correspondiente. En resumen un fichero de HTML sería de la forma:

```
<HTML>
    <HEAD>
    ...
    </HEAD>
    <BODY>
    ...
    </BODY>
</HTML>
```

<HEAD> CABECERA DEL DOCUMENTO

Esta cabecera contiene esencialmente información del documento.

<BODY> CUERPO DEL DOCUMENTO

En el cuerpo del documento se encontrarán todas las directivas HTML así como el texto, las imágenes, sonidos, tablas, listas, etc. Como descripción gráfica, podremos decir que todo lo que aparezca entre las directivas <BODY>...</BODY> será visualizado de una u otra manera en la pantalla del navegador.

Formularios en HTML

Aunque no vamos a definir en detalle otros aspectos del lenguaje, si nos detendremos un poco en los formularios, ya que serán ampliamente empleados para nuestras consultas.

Los formularios son elementos de nuestra página Web que permitirán a nuestro visitante interactuar con nuestra página. Interpretan las acciones que una persona realiza y obtienen en consecuencia determinada respuesta.

Los formularios son plantillas en las que se introducen datos para mandarlos al servidor. Cada dato es, a grandes rasgos, un par nombre/valor, donde nombre indica el campo y valor los datos introducidos en dicho campo.

En HTML un formulario es un área de código comprendida entre 2 directivas, comienza con <FORM> y finaliza con </FORM>. Cada formulario tiene asignada una acción distinta.

Estas son algunas opciones y elementos que nos dan los formularios:

<**INPUT**>

Campos de entrada por teclado.
Botones de selección.
Casillas de marca.
Botones de proceso (submit).
Botones de inicialización (reset).
Imágenes sensibles al ratón.

<**SELECT**>

Listas desplegables de valores.
Listas fijas de valores.

<**TEXTAREA**>

Ventanas de escritura libre.

En un formulario debe aparecer:

- `ACTION=URL`

  Contiene la URL del programa CGI encargado de interpretar la entrada del usuario y generar los resultados oportunos.

- `METHOD= GET | POST`

  Tiene un significado. En principio puede tomar como valor cualquiera de los métodos de transferencia de datos reconocidos por HTTP, pero en la práctica sólo se emplean 2:

a. **GET:** Añade los argumento del formulario al URL que se especifica en ACTION (usando como separador el símbolo ?), lo que da lugar a que el programa los reciba como parámetros de entrada.

b. **POST:** Envía los datos como parte de la entrada estándar.

Estructuración por frames (marcos)

Los frames son divisiones que se efectúan en la pantalla del navegador, en la que en cada frame podremos ejecutar un documento .html (.jsp en nuestro caso) independiente o no del resto de frames que hayamos definido.

Las frames es una técnica para subdividir la pantalla en diferentes ventanas. Cada una de estas ventanas se podrá manipular por separado, permitiéndonos mostrar en cada una de ellas una página Web diferente. Esto es muy útil para, por ejemplo, mostrar permanentemente en una ventana los diferentes contenidos de nuestra página, y en otra ventana mostrar el contenido seleccionado.

La forma de establecer frames es muy sencilla. En la sección <HEAD>...</HEAD> del documento se definen las áreas que ocuparán los frames que queremos poner, con ayuda de la directiva cerrada <FRAMESET>...</FRAMESET>. En esta directiva se especificarán las áreas y sus dimensiones relativas con la ayuda de atributos, como COLS y ROWS propias de <FRAMESET>...</FRAMESET>. En función de si la división de la pantalla se realiza por filas (rows) o columnas (cols), deberemos considerar cada uno de los atributos juntos o separados.

Ventajas de usar frames

- La navegación de la página será más rápida. Aunque la primera carga es igual, en sucesivas impresiones ya tendremos algunos marcos guardados, que no tendrían que volverse a cargarse.
- Como no tenemos que incluir partes de código como la barra de navegación, título, etc. crear nuevas páginas sería un proceso mucho más rápido.
- Partes de la página se mantienen fijas y eso puede ser bueno, para que el usuario no las pierda nunca de vista.
- Estas mismas partes visibles constantemente, si contienen enlaces, pueden servir muy bien para mejorar la navegación por el sitio.
- Mantienen una identidad del sitio donde se navega, pues los elementos fijos conservan la imagen siempre visible.

Inconvenientes de usar frames

- Quitan espacio en la pantalla. El espacio ocupado por los frames fijos se pierde a la hora de hacer páginas nuevas, porque ya está utilizado. En definiciones de pantalla pequeña o dispositivos como Palms, este problema se hace más patente.
- A mucha gente les disgustan pues no se sienten libres en la navegación, pues entienden que esas partes fijas están limitando su movilidad por la web.
- Los bookmarks o favoritos no funcionan correctamente en muchos casos. Si queremos incluir un favorito a una página de un frame que no sea la portada podemos encontrar problemas.
- Puede que el botón de atrás del navegador no se comporte como deseamos.
- Si queremos actualizar más de un frame con la pulsación de un enlace hay que utilizar Javascript. Además los scripts se pueden complicar bastante cuando se tienen que comunicar varios frames entre si.

3.2.2. JavaScript

INTRODUCCIÓN A JAVASCRIPT

¿Qué es JavaScript?

Mas formalmente, JavaScript, al igual que Java, es una de las múltiples maneras que han surgido para extender las capacidades del lenguaje HTML. Al ser la más sencilla, es por el momento la más extendida. Antes que nada conviene aclarar lo siguiente:

JavaScript no es un lenguaje de programación propiamente dicho. Es un lenguaje script u orientado a documento, como pueden ser los lenguajes de macros que tienen muchos procesadores de texto. Nunca podremos hacer un programa con JavaScript, tan sólo mejorar una página Web.

¿Para qué sirve JavaScript?

JavaScript sirve principalmente para mejorar la gestión de la interfaz cliente/servidor. Un script JavaScript insertado en un documento HTML ó JSP permite reconocer y tratar localmente, es decir, en el cliente, los eventos generados por el usuario. Estos eventos pueden ser el recorrido del propio documento HTML o la gestión de un formulario.

Como todos los lenguajes de programación orientados a los objetos, JavaScript establece una jerarquía de objetos, que permite definir con precisión propiedades que de otro modo resultarían indefinibles. Los principales objetos en Nestscape son: window, document, history, location y navigator. Dichos objetos están comunicados entre sí por relaciones estructurales del tipo:

Objeto1.objeto2.propiedad2

Objeto1 está en el nivel más alto, objeto2 es una propiedad de objeto1 y propiedad2 es una propiedad de objeto2.

JavaScript se compone de elementos de programación como: argumentos, gestores de eventos, funciones, literales, expresiones, métodos, objetos, operadores, propiedades, instrucciones, valores y variables.

En la práctica, JavaScript permite el enriquecimiento de documentos HTML con script más o menos complejos y más o menos útiles. La apertura de ventanas independientes de la principal del navegador es una de las peculiaridades de JavaScript, que, además de ofrecer ejemplos prácticos, permite definir y profundizar elementos conceptuales. Este es el código HTML necesario para abrir una ventana independiente de la principal del browser:

```
<HTML>
	<HEAD>
		<TITTLE>Nueva Ventana</TITTLE>
	</HEAD>
	<BODY>
		<FORM>
			<INPUT TYPE="button" VALUE="Nueva ventana"
				onClick='window.open("ventana.htm");'>
		</FORM>
	</BODY>
</HTML>
```

Para entender este script es necesario introducir el método open(). Los métodos son funciones precisas asociadas a los objetos, de los cuales piden la elaboración sin definir las características. Los métodos se reconocen fácilmente por dos paréntesis que siguen a la palabra.

OnClick es, sin embargo, uno de los principales gestores de eventos, y como tal se usa para iniciar o llamar a un script. Se usa con el tag <INPUT> de tipo interruptor (botón). Como respuesta a un click del ratón, se ejecuta el script especificado por el gestor de eventos, en este caso window.open.

JavaScript permite definir no sólo la apertura de una ventana independiente de la principal, sino también regular algunas de sus características. Para hacer esto, es necesario programar algunos parámetros que, hay que decirlo, generan sus propios efectos exclusivamente en la nueva ventana, y no en la principal del navegador.

JavaScript soporta los siguientes argumentos, en función de los cuales se pueden definir las características de la nueva ventana:

Jscript (Javascript de Microsoft)

Un lenguaje del lado del servidor es aquel que se ejecuta en el servidor web, justo antes de que se envíe la página a través de Internet al cliente. Las páginas que se ejecutan en el servidor pueden realizar accesos a bases de datos, conexiones en red, y otras tareas para crear la página final que verá el cliente. El cliente solamente recibe una página con el código HTML resultante de la ejecución de la página ASP (Active Server Pages). Como la página resultante contiene únicamente código HTML, es compatible con todos los navegadores.

Con las ASP podemos realizar muchos tipos de aplicaciones distintas. Nos permite acceso a bases de datos, al sistema de archivos del servidor y en general a todos los recursos que tenga el propio servidor. También tenemos la posibilidad de comprar componentes ActiveX fabricados por distintas empresas de desarrollo de software que sirven para realizar múltiples usos, como el envío de correo, generar gráficas dinámicamente, y un largo etc. Las versiones más modernas son el ASP.NET, que comprende algunas mejoras en cuanto a posibilidades del lenguaje y rapidez con la que funciona. ASP.NET tiene algunas diferencias en cuanto a sintaxis con el ASP, de modo que se ha de tratar de distinta manera uno de otro.

Un ejemplo:

Cuando la página HTML es un formulario que permite acceder a un anuario telefónico, se puede insertar un script que verifique la validez de los parámetros proporcionados por el usuario. Esta prueba se efectúa localmente y no necesita un acceso a la red.

Diferencias con Java.

La principal diferencia entre JavaScript y Java, es que este último es un lenguaje de programación completo, más aún, lo único que comparten es la misma sintaxis. JavaScript es un lenguaje que se integra directamente en páginas HTML.

Tiene como características principales las siguientes:

- Es interpretado (que no compilado) por el cliente, es decir, directamente del programa fuente se pasa a la ejecución de dicho programa, con lo que al contrario que los lenguajes compilados no se genera ni código objeto ni ejecutable para cada máquina en el que se quiera ejecutar dicho programa.
- Está basado en objetos. No es, como Java, un lenguaje de programación orientada a objetos (POO). JavaScript no emplea clases ni herencia, ni otras técnicas típicas de la POO.
- Su código se integra en las páginas HTML, incluido en las propias páginas.
- No es necesario declarar los tipos de variables que van a utilizarse ya que como se verá más adelante, JavaScript realiza una conversión automática de tipos.
- Las referencias a objetos se comprueban en tiempo de ejecución. Esto es consecuencia de que JavaScript no es un lenguaje compilado.

- No puede escribir automáticamente al disco duro. Por eso decimos que JavaScript es un lenguaje seguro para el entorno de internet.

UTILIZACIÓN DE JAVASCRIPT EN UN DOCUMENTO HTML.

La inserción de código en un documento HTML se realiza mediante la marca **SCRIPT** utilizando la sintaxis:

```
<SCRIPT>
Código del script
</SCRIPT>
```

Los atributos de esta marca son:

LANGUAGE="JavaScript"

Precisa el lenguaje del script. Este atributo es obligatorio en ausencia del atributo SRC.

SRC=url

El atributo SRC precisa el URL del script a insertar en el documento. Este atributo es opcional, porque el script puede insertarse directamente en un documento HTML.
Estos dos atributos pueden especificarse simultáneamente. Por ejemplo:

```
<SCRIPT LANGUAGE="lenguaje" SCR=url>
Código del script
</SCRIPT>
```

podrá especificarse para insertar en un documento un script de un lenguaje determinado y que cuyo código fuente se encuentra en un archivo especificado en un determinado url. A continuación enunciaremos algunos puntos a tener en cuenta respecto a la introducción de JavaScript en un documento HTML:

- El script insertado mediante la marca **SCRIPT** es evaluado por el cliente tras la visualización de la página HTML. Las funciones definidas no se ejecutan inmediatamente, dependen de los eventos asociados a la página.

- La inserción del script mediante la marca **SCRIPT** puede colocarse en cualquier lugar del documento HTML pero se recomienda colocarla en la cabecera, es decir, en la zona definida por el **HEAD**. De este modo, el script está definido desde el principio del documento, lo que garantiza que éste se visible en todo el documento.

- Si se definen, además del script mediante el atributo SRC, scripts en el propio documento, el cliente evaluará en primer lugar el insertado mediante el atributo SRC y seguidamente los incluidos en el documento.

- Los URL correspondientes a un JavaScript poseen generalmente la extención **.js**.
- El lenguaje JavaScript no es *case sensitive*, es decir, no distinque mayúsculas de minúsculas salvo en las cadenas de caracteres literales.

Por último, comentar otra forma de introducir scripts en documentos HTML, y es incluir estos script como controladores de eventos de algunas marcas, como pueden ser la marcas de imágenes, anclas, links, botones, etc. Veamos a continuación un ejemplo:

```
<A HREF="index.htm" OnClick="alert('ir al índice')">
Ir al índice
</A>
```

Como puede verse, dentro de la marca, como atributo de esta, se pone un controlador de eventos y después del signo igual y entre comillas se incluye el código de JavaScript. Ahora bien, también es posible llamar a una función del HEAD del documento. Se recomienda esta segunda opción ya que es una manera más limpia y clara de escribir páginas. Se conseguiría lo mismo que en el ejemplo anterior de esta forma:

```
<HEAD>
  <SCRIPT LANGUAGE="JavaScript">
            function alerta()
            {
                 alert(" Ir al índice");
            }
  </SCRIPT>
  ...
</HEAD>

<BODY>
      <A HREF="index.jsp" OnClick="alerta()"> Ir al índice </A>
...
</BODY>
```

LAS VERSIONES DE JAVASCRIPT

JavaScript ha sido creado gracias a una estrecha colaboración entre Netscape y Sun Microsystem, y resulta obvio, por tanto, que la comprensión de semejante lenguaje no pueda prescindir de Java. Hay que precisar que JavaScript es algo muy distinto de Java. Ambos lenguajes están dirigidos a los objetos, pero mientras Java se usa para crear aplicaciones autónomas, o applet, JavaScript se interpreta con el código HTML (del que forma parte integrante, y sin el cual no puede existir), sin necesidad de máquinas virtuales o conocimientos profundos de modelos orientados a los objetos. Sendos lenguajes tienen en común parte de la sintaxis y de la estructura.

La versión 1.0 de JavaScript nació con el Netscape Navigator 2.0. Posteriormente, surgieron las versiones 1.1 y 1.2 de JavaScript con las versiones 3 y 4 del Netscape. También existe una versión 1.3, introducida en la versión 4.07 del Netscape Navigator. Esta

versión es una pequeña revisión de la 1.2 creada para ajustarse al estándar internacional ECMA que regula el lenguaje JavaScript.

En cuanto a Microsoft Internet Explorer en su versión 3.0 interpreta JScript, que es muy similar a JavaScript 1.0 pero con algunas incompatibilidades. Pero ya su versión 4.0 soporta sin ningún problema, la versión 1.1 de JavaScript.

SINTAXIS.

JavaScript, como cualquier lenguaje de programación, por sencillo que sea, posee una sintaxis propia del lenguaje, sin embargo, no vamos a hacer hincapié en ella en esta memoria, pues no es el objetivo de nuestro trabajo. Si el lector quisiera profundizar en la materia, le remitimos a la búsqueda en Internet, en la que encontrará multitud de manuales y referencias sobre el tema.

EJEMPLOS DE USO.

Veremos ejemplos del uso de JavaScript en secciones posteriores de esta memoria, en el momento que hablemos de las funciones javascript concretas utilizadas.

3.2.3 JSP (Java Server Pages)

INTRODUCCIÓN

¿Qué es JSP?

Java Server Pages (JSP, para abreviar) es una tecnología basada en Java que simplifica el desarrollo de páginas web con contenido dinámico. Con JSP, los diseñadores web pueden incorporar elementos dinámicos dentro de la página utilizando tanto porciones de código Java incrustadas, como unas cuantas etiquetas (<% ... *código JAVA (script)*... %>). Así, las páginas JSP tienen el aspecto de una página tradicional HTML, a la que se le ha introducido parte de código Java junto con unas etiquetas. De esta forma, cuando una página es solicitada por un usuario y procesada por un servidor HTTP, el código HTML pasará directamente al usuario, mientras que las porciones de código Java serán ejecutadas en el servidor cuando la solicitud haya sido recibida, para generar el contenido dinámico de la página. Cuando el usuario acceda al código de la página que le llega sólo verá HTML, sin poder acceder al código JSP subyacente.

Evolución tecnológica

El hecho de que una página contenga contenido dinámico, exige que el servidor web realice algún trabajo de procesado que en el caso de páginas estáticas no era necesario. La forma de realizar este procesado ha ido cambiando con el tiempo, y así veremos ahora un pequeño resumen de cómo ha ido evolucionando.

Inicialmente la generación de contenido dinámico se hacía fuera de los servidores. Cuando llegaba una solicitud, esta era procesada en el servidor, y cuando era necesario se llamaba a un proceso, fuera del servidor, que generaba el contenido dinámico y lo devolvía al

servidor. Los modelos basados en el *Common Gateway Interface* (CGI) seguían esta idea. Sus principales problemas eran: necesidad de *overhead* para la comunicación entre proceso y servidor, consumo de recursos de sistema por parte del proceso, etc.

Posteriormente se optó por sistemas que introdujeran porciones de código de lenguaje tradicionales incrustados dentro de la página usando etiquetas, de forma que el estilo de la sintaxis fuera más consistente y sencillo. Este modelo lo han seguido entre otros: *Microsoft Active Server Pages (ASP), Server Side JavaScript (SSJS), Java Server Pages (JSP)* y otros.

En JSP el lenguaje que se utilizará para la generación de contenido dinámico es, típicamente, Java, y provee un conjunto de etiquetas que interactúan con objetos Java en el servidor de forma que no es estrictamente necesario que aparezca código Java en la página.

Beneficios de usar JSP

Al estar basado en Java, presenta las ventajas que este lenguaje ofrece con respecto a la portabilidad entre plataformas y las derivadas de la orientación a objetos de este lenguaje.

Realización.

Las peticiones de páginas JSP son normalmente implementadas mediante *servlets*, de forma que el contenedor *servlet*, al que llamaremos contenedor JSP, maneja múltiples solicitudes a la vez, requiriendo menor *overhead*, y por tanto requiriendo menos recursos. Esto hace que JSP sea mucho más eficiente que otros modelos como los programas CGI .

Componentes reutilizables.

Esta característica deriva de la orientación a objetos de Java. JSP permite implementar contenido dinámico incluyendo código Java directamente en la página.

SINTAXIS DE JSP.

En primer lugar hay que decir que el contenedor JSP acepta las etiquetas HTML, de forma que la solicitud a un contenedor JSP local del siguiente fichero, al que habremos dado su correspondiente extensión (normalmente .jsp), sería perfectamente procesado, y por lo tanto sería un fichero JSP válido.

```
<HTML>
    <BODY>
      ¡Hola, mundo!
    </BODY>
</HTML>
```

Veamos ahora un ejemplo más interesante:

```
<HTML>
    <BODY>
       <% String visitante = request.getParameter("nombre");
        if (visitante==null) visitante = "mundo"; %>
        ¡Hola, <%= visitante%>!
    </BODY>
</HTML>
```

Sin entrar en detalles, este fichero JSP declara una variable Java *String* llamada *visitante*, y luego la intenta inicializar con un parámetro que busca en la solicitud HTTP. En caso de no encontrarlo, le da el valor *mundo*, y posteriormente inserta el valor de esta variable en la salida HTML de esta página.

Con este simple ejemplo ya hemos visto la característica central de las páginas con contenido dinámico: dependiendo del valor de los parámetros en la solicitud, el resultado de la misma es distinto. Otro aspecto importante es el hecho de que el navegador no tiene constancia del código JSP de la página. Este código se ejecuta en el contenedor JSP, generando contenido HTML, y devolviéndolo completo al servidor HTTP, que lo devuelve al navegador sin dejar ninguna constancia de toda esta labor.

Formato de las etiquetas.

Una vez que hemos visto unos ejemplos que nos muestran la forma que toman los documentos JSP, llega el momento de clasificar las etiquetas que se utilizan en este lenguaje. Estas etiquetas quedan englobadas en dos tipos de formato: las etiquetas orientadas a *scripting* y las basadas en XML.

1.- Orientadas a *scripting*.

Son etiquetas derivadas del ASP (*Microsoft Active Server Page*) y toman el formato de empiezar por ´<%´ y terminar por ´%>´. Un carácter especial puede aparecer después del ´<%´ inicial, tal como: ¡ , = , @ y sirven para añadir significado a la etiqueta.

```
<%! double radio=7.8; %>
<%@page include file="copyright.html" %>
```

2.- Basadas en XML..

El segundo tipo de etiqueta viene de la sintaxis utilizada en XML y sus convenciones. Esta sintaxis es muy parecida a la de HTML, pero añadiendo unas pocas reglas tales como diferenciar entre mayúsculas y minúsculas.

Empiezan por < y terminan por />.

```
<jsp:usebean id="login" class="com.taglib.wdjsp.fundamentals"/>
```

JSP soporta el anidamiento de etiquetas una dentro de otra. Así, cualquier etiqueta JSP puede estar dentro de las etiquetas HTML del documento, permitiendo generar contenido dinámico de las propias etiquetas. Igualmente, JSP permite contener, bajo ciertas restricciones, etiquetas JSP dentro de otras etiquetas JSP. Aquí tenemos un ejemplo:

```
<jsp:setProperty name="login" property="visitantes" value="<% visitasPrevias + 1%>"/>
```

EJECUCIÓN DE JSP

Con lo que hemos visto, deberíamos tener una idea general de qué es lo que JSP puede hacer. Ahora veremos cómo lo hace.

Requisitos para JSP.

Lo primero que necesitamos para trabajar con JSP es un servidor web. Cuando hablamos de servidor, nos referimos tanto a una parte *hardware* como a una parte *software*. La parte *hardware* es normalmente un ordenador accesible desde Internet o desde una red corporativa (el servidor de la facultad de físicas), mientras que la parte *software* es un servidor HTTP ejecutándose en el *hardware*

Además del servidor HTTP, es necesario un *sofware* que implemente un contenedor JSP. Muchos servidores incluyen este contenedor. En otros, será necesario incluir este contenedor. Actualmente, la mayoría de proveedores de servidores HTTP ofrecen API (*Application Programming Interfaces*) para incorporar la generación de contenido dinámico a la funcionalidad HTTP.

Una vez que tenemos el servidor instalado y configurado, añadiremos los ficheros JSP a la jerarquía normal de documentos del servidor, y se distinguirán del resto por su extensión, normalmente .jsp aunque no siempre es esta. Las clases Java, ya compiladas, que son referidas por nuestros documentos JSP necesitan ser instaladas en el path de las clases Java que utiliza el contenedor Java.

¿Cómo trabaja JSP?

El proceso de ejecución de un documento JSP empieza con la solicitud del mismo. Estas solicitudes están indicadas por el URL que emplea una extensión especial, que ya dijimos que generalmente era *.jsp* pero podría ser otra.

Servlets: La mayoría de las implementaciones de JSP están basadas en los *servlets*. Por ello, el primer paso para comprender cómo trabaja JSP, es comprender cómo trabajan los *servlets*.

De algún modo, son programas que se ejecutan en un servidor Web y construyen páginas Web. Construir páginas Web al vuelo es útil (y comúnmente usado) por un número de razones:

- **La página Web está basada en datos enviados por el usuario**. Por ejemplo, las páginas de resultados de los motores de búsqueda se generan de esta forma, y los programas que procesan pedidos desde sites de comercio electrónico también.
- **Los datos cambian frecuentemente**. Por ejemplo, un informe sobre el tiempo o páginas de cabeceras de noticias podrían construir la página dinámicamente, quizás devolviendo una página previamente construida y luego actualizándola.

Los *servlet*s son programas basados en Java, análogos a los programas CGI, implementados mediante un contenedor *servlet* asociado a un servidor HTTP. El fundamento de los *servlets* es el siguiente; un conjunto de URLs son configurados para ser administrados por el contenedor *servlet*, de forma que siempre que llegue una solicitud para una de estos URLs en el servidor, este lo envía al contenedor *servlet* para que lo procese.

La forma de enviarlo es creando un objeto Java que empaquete todos los datos de la solicitud. Un objeto Java también es creado representando la respuesta. Ambos objetos tendrán sus métodos de acceso, de esta forma, el contenedor *servlet* accede a los datos de la solicitud para realizar las operaciones necesarias sobre los mismos y así construir la respuesta. El código HTML generado como respuesta es escrito en la cadena de salida asociada al objeto respuesta, y este objeto es enviado al servidor HTTP, el cual la devuelve al navegador que hizo la solicitud en primer lugar. En el caso de que existan múltiples solicitudes para un *servlet*, están son administradas ejecutando cada llamada a los métodos de los *servlet*s en diferentes threads.

JavaServer Pages: El componente principal de una implementación de JSP basada en *servlets* es un *servlet* especial llamado compilador de página. El contenedor está configurado para llamar a este *servlet* siempre que llega una solicitud a una página JSP. Es este compilador de página y su clase Java asociada el que vuelve al contenedor *servlet* en un contenedor JSP.

El procedimiento es el que sigue. Cuando llega al servidor HTTP una solicitud de una página JSP, esta solicitud es enviada al contenedor JSP, el cual invoca al compilador de página para que se encargue de la misma. El compilador analiza el contenido de la página buscando etiquetas JSP, traduciendo su contenido en el código Java equivalente que, al ser ejecutado, generará el contenido dinámico. Mezclando el contenido estático de la página original junto con el código Java del contenido dinámico, se generará un *servlet* con sus métodos de servicio. Una vez que todo el código del *servlet* ha sido construido, el compilador de página llama al compilador Java para compilar este código y añadir el fichero de clase Java resultante al directorio apropiado en el path de las clases del contenedor JSP. Todo este proceso sólo se realiza la primera vez que se solicita una página JSP, el resto de solicitudes son remitidas directamente al *servlet* compilado. Así cuando se llama a una página JSP el

compilador de página invoca a este *servlet* para generar la respuesta para la solicitud original.

Resumiendo, podemos decir que las solicitudes del navegador llegan al servidor HTTP y las páginas JSP son enviadas al *servlet* compilador de páginas que corre en el contenedor JSP. Si el *servlet* para la página actual está actualizado lo genera y compila, cargándolo en el contenedor *servlet*. En caso contrario el control es transferido al *servlet* de la página JSP que se encarga de manejar la solicitud generando la respuesta y enviándola al servidor HTTP el cual la remitirá al navegador.

Ventajas de los Servlets sobre el CGI "Tradicional"

Los Servlets Java son más eficientes, fáciles de usar, más poderosos, más portables, y más baratos que el CGI tradicional y otras muchas tecnologías del tipo CGI.

- **Eficiencia**. Con CGI tradicional, se arranca un nuevo proceso para cada solicitud HTTP. Si el programa CGI hace una operación relativamente rápida, la sobrecarga del proceso de arrancada puede dominar el tiempo de ejecución. Con los Servlets, la máquina Virtual Java permanece arrancada, y cada petición es manejada por un thread Java de peso ligero, no un pesado proceso del sistema operativo. De forma similar, en CGI tradicional, si hay N peticiones simultáneas para el mismo programa CGI, el código de este problema se cargará N veces en memoria. Sin embargo, con los Servlets, hay N threads pero sólo una copia de la clase Servlet. Los Servlet también tienen más alternativas que los programas normales CGI para optimizaciones como los cachés de cálculos previos, mantener abiertas las conexiones de bases de datos, etc.

- **Conveniencia**. Los Servlets tienen una gran infraestructura para análisis automático y decodificación de datos de formularios HTML, leer y seleccionar cabeceras HTTP, manejar cookies, seguimiento de sesiones, y muchas otras utilidades.

- **Potencia**. Los Servlets Java nos permiten fácilmente hacer muchas cosas que son difíciles o imposibles con CGI normal. Por algo, los servlets pueden hablar directamente con el servidor Web. Esto simplifica las operaciones que se necesitan para buscar imágenes y otros datos almacenados en situaciones estándar. Los Servlets también pueden compartir los datos entre ellos, haciendo las cosas útiles como almacenes de conexiones a bases de datos fáciles de implementar. También pueden mantener información de solicitud en solicitud, simplificando cosas como seguimiento de sesión y el caché de cálculos anteriores.

- **Portable**. Los Servlets están escritos en Java y siguen un API bien estandarizado. Consecuentemente, los servlets escritos, digamos en el servidor I-Planet Enterprise, se pueden ejecutar sin modificarse en Apache, Microsoft IIS, o WebStar. Los Servlets están soportados directamente o mediante plug-in en la mayoría de los servidores Web.

- **Barato**. Hay un número de servidores Web gratuitos o muy baratos que son buenos para el uso "personal" o el uso en sites Web de bajo nivel. Sin embargo, con la

excepción de Apache, que es gratuito, la mayoría de los servidores Web comerciales son relativamente caros. Una vez que tengamos un servidor Web, no importa el coste del servidor, añadirle soporte para Servlets (si no viene preconfigurado para soportarlos) es gratuito o muy barato.

Ventajas de JSP frente a:

- **Active Server Pages (ASP)**. ASP es una tecnología similar de Microsoft. Las ventajas de JSP están duplicadas. Primero, la parte dinámica está escrita en Java, no en Visual Basic, otro lenguaje específico de MS, por eso es mucho más poderosa y fácil de usar. Segundo, es portable a otros sistemas operativos y servidores Web.

- **Los Servlets**. JSP no nos da nada que no pudiéramos en principio hacer con un servlet. Pero es mucho más conveniente escribir y modificar HTML normal que tener que hacer un billón de sentencias println que generen HTML. Además, separando el formato del contenido podemos poner diferentes personas en diferentes tareas: nuestros expertos en diseño de páginas Web pueden construir el HTML, dejando espacio para que nuestros programadores de servlets inserten el contenido dinámico.

- **Server-Side Includes (SSI)**. SSI es una tecnología ampliamente soportada que incluye piezas definidas externamente dentro de una página Web estática. JSP es mejor porque nos permite usar servlets en vez de un programa separado para generar las partes dinámicas. Además, SSI, realmente está diseñado para inclusiones sencillas, no para programas "reales" que usen formularios de datos, hagan conexiones a bases de datos, etc.

- **JavaScript**. JavaScript puede generar HTML dinámicamente en el cliente. Esta es una capacidad útil, pero sólo maneja situaciones donde la información dinámica está basada en el entorno del cliente. Con la excepción de las cookies, el HTTP y el envió de formularios no están disponibles con JavaScript. Y, como se ejecuta en el cliente, JavaScript no puede acceder a los recursos en el lado del servidor, como bases de datos, catálogos, información de precios, etc.

3.2.4. CSS (Cascade Style Sheets)

INTRODUCCIÓN

Pensemos en una historia:

"...Erase una vez un diseñador de sitios web al que un cliente le encarga un proyecto en el campo de las casas rurales. Tras días y días de trabajar sin descanso, presentó un precioso diseño: Unas columnas azul oscuro, sobre un fondo rojo, que combinaban con unos textos de enlace en Verdana y color amarillos, para distinguirlos del texto normal (Arial en verde). Así más de 150 páginas. Llegado el día de la presentación fue alegremente, contento con su trabajo, donde el cliente y éste le dijo "Mmmm... Me gustaría más el fondo blanco y las letras normales en morado. Y si pudiera ser cambiar también el color de las columnas..."

Entonces el diseñador pensó:

"yo lo mato; a reformar 150 páginas... AAAAGH!!!"

Pues para este tipo de situaciones, además de la estandarización y la portabilidad, surgen las CSS (Cascade Style Sheets).

Breve historia

En los orígenes de la Web y en sus primeras versiones, HTML era un lenguaje fácil de aprender y muy reducido en cuanto a sus tags y estructura. Estamos hablando de los años 1990 al 1993. Todo cambió cuando empezaron a surgir los primeros navegadores que eran capaces de representar recursos gráficos como añadido a la información textual.

Así, el número de sitios web comenzó a crecer y con él, el número de tags que la especificación HTML contemplaba. El objetivo era construir sitios web cada vez más atractivos visualmente hablando, con lo que HTML debía incluir nuevos tags destinados a conseguir diversos efectos visuales.

Con todos estos cambios que la web había sufrido, nos encontramos con que un lenguaje que en sus inicios había sido "orientado a la estructura", ahora estaba totalmente "orientado a la visualización" (HTML 4 es la más viva representación de esta realidad). Encontramos tags como <B>, <U> o <I> que definen estilos de visualización sin aportar nada a la estructura del documento representados.

Otro aspecto importante y que condiciona totalmente la estructura del documento es el uso del tag FONT. Con el uso de este tag podemos hacer que una zona que corresponde a la cabecera o título de una página, y que debería expresarse con un H1, pase ahora a estar definida mediante el tag FONT. **Con este cambio se pierde totalmente la estructura del documento.**

La realidad ahora es que el mayor número de los sites realizados con HTML 4 consiguen que el volumen de información de visualización sea muy superior al de la información verdaderamente relevante.

Motivos por los que no podemos permitir que nuestros documentos publicados en la web pierdan su estructura son:

- La indexación por los buscadores es mucho más complicada
- Se reduce la accesibilidad. Actualmente existen aplicaciones que permiten la lectura de páginas web como ayuda a los discapacitados (persona ciegas o con otras discapacidades). Si una persona discapacitada intenta acceder a una página sin una mínima estructuración, el resultado puede ser lamentable.
- La estructura de la página y la información contenida en la misma es mucho más sencilla de mantener. Actualmente, ciertos aspectos del código HTML pueden hacer que una misma página tenga visualizaciones distintas en distintos navegadores. Estos errores de diseño son difícilmente depurables cuando la página contiene una estructura de tags complicada y sin ninguna estructuración. Por otra parte, un

cambio en un tipo de fuente supone el rediseño de todas las páginas de un site al tener que sustituir todos los valores para el tag FONT.

Todos estos problemas han sido seguidos muy de cerca por el W3C, el cual comenzó a trabajar en 1995 en CSS.

Ventajas de las CSS frente a HTML 4.

- **Estilo enriquecido**. CSS permite la creación de documentos visualmente mucho más ricos que lo que HTML nunca permitirá. No en vano CSS está pensado única y exclusivamente para asistir al diseñador a la hora de dar estilo a un documento estructurado.

- **Fácil de utilizar**. La utilización de hojas de estilo CSS hace que el diseñador pueda reducir sustancialmente su carga de trabajo al diseñar todo un site. Esto se debe a que CSS es capaz de centralizar ciertos efectos visuales que plasmemos en diversas secciones del site, en lugar de tenerlos diseminados por páginas y páginas del site.

- **Reutilización en múltiples páginas**. Una hoja de estilo que recoja aspectos visuales comunes a varias páginas puede ser reutilizada en cualquier sección del site aprovechando dichos efectos ya definidos. De esta manera es sencillo generar un estilo general del web y mantenerlo así consistente para todas las páginas. Así, si deseamos modificar un estilo que es común a todo el site, sólo necesitaríamos modificar una línea de nuestro fichero CSS (con la aproximación clásica que ofrece HTML, deberíamos modificar todas y cada una de las páginas).

- **Reduce el tamaño del código HTML enviado**. Al centralizar los estilos ya no es necesario la utilización de tags como FONT en todos los documentos del site. De esta manera se reduce considerablemente el tiempo de carga de una página.

- **Nos prepara par el futuro**. Debemos ser conscientes que muchos tags como FONT, BASEFONT, U, STRIKE, S, CENTER, han sido marcados por el W3C como "*deprecated*", es decir, que se desaconseja su uso ya que serán eliminados en un futuro de la especificación. De igual manera HTML retornará progresivamente a sus orígenes, volviendo a ser un lenguaje para la estructuración de información.

En definitiva, las CSS suponen un gran avance en la autoría de sitios Web. Mejoran las posibilidades de diseño y presentación de documentos en la red, facilitando además su mantenimiento, ya se trate de un único archivo HTML, o de grandes sitios, con multitud de páginas. La filosofía de las CSS responde a la idea de ***separar al máximo forma y fondo***. Las páginas, idealmente, tendrán únicamente etiquetas html con información acerca de la estructura y contenido del documento (párrafos, cabeceras, listas etc) sin ninguna información sobre la apariencia del documento. Esta apariencia, la forma en que ese documento debe ser visualizado, se dicta por una serie de reglas, que se definen separadamente (en una sección separada de la página o incluso en un archivo separado), de forma que podemos cambiar la forma en que el documento es visualizado sin tocar ni una sola línea de contenido ni cambiar las etiquetas html, simplemente introduciendo unos pocos cambios en la definición de estilo.

Las reglas de estilo se escriben y leen en un lenguaje inteligible, similar al lenguaje cotidiano (naturalmente, en inglés), utilizando términos habituales en el área de diseño gráfico, lo que permite un rápido aprendizaje.

Las hojas de estilo en cascada (CSS1) responden a un standard definido por la organización W3C [traducción], y este standard, al menos en su nivel 1, "se supone" ha de funcionar correctamente con las versiones mas recientes de los navegadores gráficos mas populares, por lo que hace tiempo que ha dejado de tener carácter experimental.

Antes de la llegada de las hojas de estilo, los autores tenían un control limitado sobre la representación. HTML 3.2 incluía un número de atributos y elementos que ofrecían control sobre la alineación, el tamaño de la fuente y el color del texto. Además los autores utilizaban las tablas y las imágenes como medio de organizar la presentación de sus páginas. El tiempo relativamente largo que necesitan los usuarios para actualizar sus navegadores hará que estas características sigan siendo usadas durante algún tiempo. Sin embargo, al ofrecer las hojas de estilo mecanismos de presentación más potentes, el World Wide Web Consortium declarará obsoletos en el futuro muchos de los elementos y atributos de presentación del HTML.

¿Que es una hoja de estilo CSS? Primeros Conceptos

Una 'hoja de estilo' es una plantilla o conjunto de instrucciones que dicta al navegador como debe mostrar el contenido de una o varias páginas Web. Cualquier cambio en la plantilla de estilo se refleja en un cambio inmediato en la apariencia de las páginas relacionadas, sin necesidad de modificar físicamente estas.

Formulada la primera recomendación CSS1 en Diciembre de 1996, las hojas de estilo traen a la edición Web una relativa libertad de diseño, ampliando el control sobre la apariencia y posicionamiento de todos los elementos de la página, efectos tipográficos, fuentes, colores, backgrounds, compatibilidad entre navegadores y sistemas operativos.

Navegadores que admiten CSS1

CSS 1 alcanzó el status de *recomendación* en el año 1996. Las reglas CSS1 tienen un *soporte adecuado* en prácticamente todos los navegadores modernos: cualquier ordenador basado en Gecko (Mozilla, Firefox), Opera, Internet Explorer, por citar los mas conocidos.

Las reglas CSS 2 alcanzaron el status de recomendación en el año 1998. Como conjunto, no son reconocidas por ningún navegador, aunque los de última generación, como Mozilla a partir de su versión 1.0, Firefox, Opera 7 o internet Explorer 6 si soportan algunas de sus reglas.

Sintaxis y estructura de las CSS.

Como en puntos anteriores, no vamos a ser exhaustivos en la definición del lenguaje de las hojas de estilo, sin embargo, si vamos a presentar su estructura básica y el significado de sus secciones.

Diseñar hojas de estilo es bastante sencillo, sólo hay que saber cómo escribirlas y seguir algunas normas básicas. Por ejemplo, para marcar el color de la etiqueta <H1> como blanco (#FFFFFF) se pone:

```
<STYLE>
H1 { color: #FFFFFF }
</STYLE>
```

Toda definición de estilo tiene 2 partes:

- Un **selector** (H1 en el ejemplo), le dice al navegador que etiqueta es a la que tiene que dar formato.
- Una **declaración** (color: #FFFFFF) que le dice qué formato es el que tiene que aplicar.

La declaración a su vez se divide en dos partes:

- Una **propiedad** (color en el ejemplo)
- Un **valor** (#FFFFFF en el ejemplo)

Cualquier etiqueta HTML puede funcionar como selector. Hay aproximadamente unas 50 propiedades en CSS1 que pueden ser aplicadas: para una lista completa de sus declaraciones y propiedades, nos remitimos a W3C.

Otro sistema para definir CSS es usar clases personalizadas. La norma para una clase personalizada tiene este aspecto:

```
<STYLE>
.textrojo { FONT-SIZE: 12px; COLOR: red }
</STYLE>
```

Para aplicar este estilo a una etiqueta, llamaremos a este estilo desde el HTML. Por ejemplo, si se quiere usar en un encabezado y en el párrafo siguiente, escribiremos:

```
<H1 class="textrojo">este texto sería rojo y de 12 píxeles de altura</H1>
<P class="textrojo">Y lo que fuera en este párrafo, también</P>
```

Hay varias pseudo-clases en la especificación CSS1, pero las más conocidas y las que utilizamos en nuestro caso, son las relacionadas con la etiqueta <A> y que son las que se utilizan para los efectos generales de los enlaces:

```
a:link { color: red }           /* un enlace no visitado*/
A:visited { color: blue }       /* enlace visitado */
A:hover { color: black }        /* raton encima del enlace*/
A:active { color: lime }        /* enlace activo */
```

4. Diseño e implementación

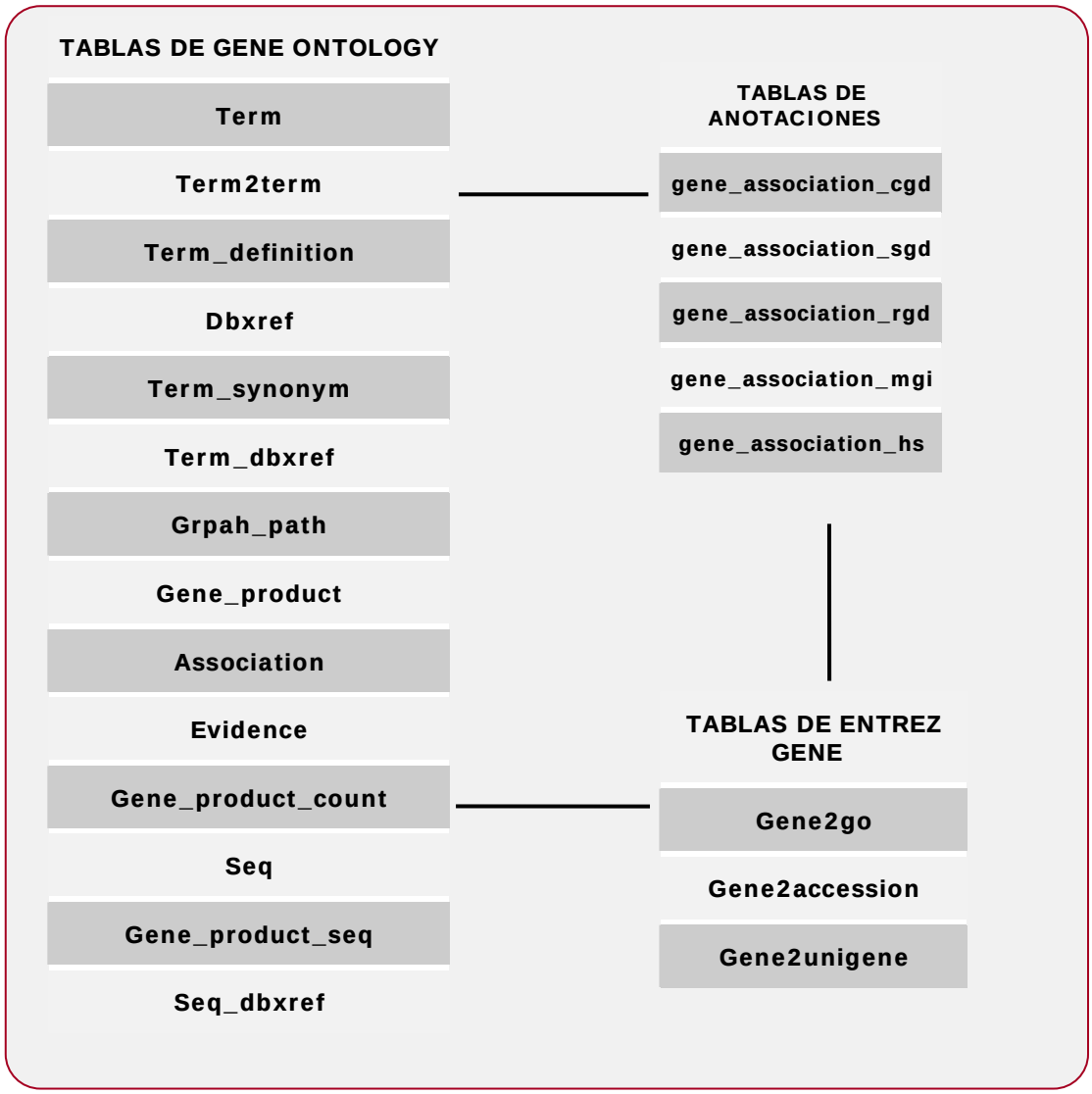
4.1. Diseño de la Base de Datos

4.1.1 Estructura de la base de datos Proyecto Genome

La base de datos Proyecto Genome comprende muchas tablas de diferentes bases de datos disponibles de forma pública para su consulta, así como otras creadas expresamente para este proyecto. Podemos establecer una estructura de la base de datos del Proyecto Genome de la forma siguiente:

1. Tablas pertenecientes a la base de datos de Gene Ontology
2. Tablas pertenecientes a la base de datos Entrez Gene
3. Tablas de las anotaciones de varios organismos

Un esquema básico de la estructura de la base de datos sería:



La base de datos de Gene Ontology se construye directamente a partir de los ficheros de datos de las tres ontologías, disponibles en la página web del proyecto. Se utiliza únicamente para consultar datos, ya que las nuevas entradas o modificaciones se hacen directamente sobre dichos archivos de datos. Las tablas pertenecientes a la base de datos de Gene Ontology que se han incorporado al Proyecto Genome son:

Tabla	Nombre de la tabla	Descripción
1.	**Term**	Términos GO
2.	**Term2term**	Relaciones entre los términos GO
3.	**Term_definition**	Definición de cada término
4.	**Dbxref**	Identificadores en otras bases de datos externas
5.	**Term_synonym**	Sinónimos para cada termino
6.	**Term_dbxref**	Enlaces de un término a otras base de datos externas
7.	**Grpah_path**	Cierre transitivo en el grafo
8.	**Gene_product**	Proteína, gen o entidad anotada
9.	**Association**	Relación entre el termino GO y los elementos de la tabla Gene_product
10.	**Evidence**	Tipo de anotación y referencia a la tabla Association
11.	**Gene_product_count**	Numero de productos genéticos por término GO de forma recursiva
12.	**Seq**	Secuencia biológica
13.	**Gene_product_seq**	Enlace entre un producto y una secuencia biológica
14.	**Seq_dbxref**	Enlaces a otras bases de datos externas para una secuencia

La base de datos GO está disponible de forma gratuita para descargada en la página http://www.godatabase.org/dev/database/.

Las diferentes versiones disponibles (mensual, semanal, diaria, clasificadas en función de la cantidad de tablas y datos que incluyen), pueden ser descargadas en diferentes formatos: RDF-XML, OBO-XML, OWL, SQL y TABLES (no es mas que un directorio que contiene por separado los datos y el código sql para cada tabla).

4.1.3 Las tablas de la base de datos Entrez Gene

La base de datos Entrez Gene proporciona resultados e informes sobre la función y la estructura de cientos de genes. Cada uno de las entradas contiene enlaces a otras fuentes (referencias literarias u otras bases de datos). Entrez Gene contiene información genética de muchos organismos diferentes, incluidos virus, bacterias, plantas, humanos y otros animales.

Entrez Gene fue creada en sustitución de la conocida base de datos LocusLink en el año 2004 de forma que para todos aquellos genomas que habían sido integrados en LocusLink, el identificador de Entrez Gene siguió siendo el mismo. Entrez gene proporciona una asociación entre sus identificadores y los términos GO, permitiendo trabajar a la vez con ambas bases de datos.

Las tablas pertenecientes a la base de datos Entrez Gene que se han incorporado al Proyecto Genome son:

Tabla	Nombre de la tabla	Descripción
1.	**gene2go**	Contiene los términos GO que han sido asociados con genes de la base de datos Entrez Gene. Se genera procesando ciertos ficheros de la base de datos de Gene Ontology y comparando el campo DB_Object_ID.
2.	**gene2accesion**	Contiene secuencias de bases de datos colaboradoras con el proyecto como SwissProt y RefSeq.
3.	**gene2unigene**	Relaciona los Unigene IDs con los Gene IDs. Unigene es un sistema experimental para división automática de secuencias de GenBank dentro de un conjunto no redundante de genes orientados a clusters.

La base de datos Entrez Gene pertenece y es mantenida por el Centro Nacional de Información Biotectonológica (NCBI), parte de la Biblioteca Nacional de Medicina de Estados Unidos. Esta es la web de Entrez Gene en el Centro Nacional de Información Biotectonológica: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=gene y la base de datos está disponible de forma gratuita para descargada en la página ftp://ftp.ncbi.nlm.nih.gov/gene/.

4.1.4 Las tablas de anotaciones

Estas tablas contienen la información que relaciona cada término GO con los elementos genéticos correspondientes. Las bases de datos que colaboran en el proyecto GO anotan los productos genéticos (o genes) con términos GO, incluyendo en cada anotación información referente al tipo de prueba que aportan para apoyar la anotación. Cada producto genético puede tener una o varias funciones moleculares, ser usado en uno o varios procesos biológicos y estar asociado con uno o varios componentes celulares.

4.1.5 Estructura de las tablas de anotaciones de GO

Todas las tablas de anotaciones creadas y utilizadas en este proyecto siguen la misma estructura y todas siguen la misma nomenclatura: **go_association_ABC**, donde ABC sería el nombre organismo anotado. Cada tabla consta de los siguientes campos:

Columna	Nombre de la columna	Tipo	Ejemplo de contenido
1.	**DB**	Varchar(55)	SGD
2.	**DB_Object_ID**	Varchar(255)	S000000296
3.	**DB_Object_Symbol**	Varchar(128)	PHO3
4.	**NOT_Qualifiers**	Varchar(25)	contributes_to
5.	**GOid**	Varchar(255)	GO:0003993
6.	**DBReference**	Varchar(255)	SGD_REF:S000047763|PMID:2676709
7.	**Evidence**	Varchar(8)	IMP
8.	**With_From**	Varchar(255)	SGD:S000000126|SGD:S000004660
9.	**Aspect**	Varchar(55)	F
10.	**DB_Object_Name**	Varchar(55)	acid phosphatase
11.	**Synonym**	Varchar(255)	YBR092C
12.	**DB_Object_Type**	Varchar(255)	gene
13.	**taxon**	Varchar(55)	Taxon:4932
14.	**Date**	Int(11)	20010118

15.	**Assigned_by**	Varchar(55)	SGD

DESCRIPCIÓN DE LOS CAMPOS

La siguiente tabla muestra una breve descripción de los campos que componen cada una de las tablas de anotaciones del proyecto:

Columna	Nombre de la columna	Descripción del contenido
1.	**DB**	Se refiere a la base de datos que proporciona el fichero de las anotaciones
2.	**DB_Object_ID**	Identificador de la anotación
3.	**DB_Object_Symbol**	Símbolo asociado al identificador. Si se anota un gen, se usará el nombre del mismo.
4.	**NOT_Qualifiers**	Toma uno de los siguientes valores: NOT, contributes_to, colocalizes_with. Permite diferenciar entre anotaciones del mismo tipo.
5.	**Goid**	identificador GO del gen anotado.
6.	**DBReference**	Es el valor que indican la fuente de la anotación. Puede ser una referencia a un libro, un artículo, una base de datos, etc.
7.	**Evidence**	Representa la fuente de la anotación. Como ya se explicó anteriormente, puede tomar uno de los siguientes valores: IMP, IGI, IPI, ISS, IDA, IEP, IEA, TAS, NAS, ND, IC, RCA.
8.	**With_From**	Puede tomar uno de los siguientes valores: DB:gene_symbol, DB:gene_symbol, DB:gene_id, DB:protein_name, DB:sequence_id, GO:GOid. Puede representar otro gen con el que el gen anotado interactúa o al que es similar.
9.	**Aspect**	Representa si se trata de un proceso biológico(P), una función molecular(F) o de un componente celular(C).
10.	**DB_Object_Name**	Nombre del gen anotado
11.	**Synonym**	Sinónimos del producto anotado
12.	**DB_Object_Type**	Representa el tipo de organismo anotado. Puede tomar uno de los siguientes valores: gene, transcript,

		protein, protein_structure, complex
13.	**taxon**	Identificador o identificadores de la taxonomía.
14.	**Date**	Fecha de la anotación con el formato YYYYMMDD
15.	**Assigned_by**	Es similar al primer campo. Representa la base de datos que aporta la anotación. Por defecto, toma el mismo valor que el campo DB, pero será diferente si la anotación es hecha por una base de datos pero incorporada en otra

4.1.6 Mantenimiento y actualización de la BD Proyecto Genome

INTRODUCCIÓN A CRONTAB

Como ya se ha comentado anteriormente, uno de los principales objetivos del Proyecto Genome era mantener la base de datos actualizada, ya que los datos que se usan provienen de bases de datos y archivos que están en constante cambio.

Para ello se empleo la herramienta Crontab de Linux, que es un programa que permite ejecutar programas o scripts con la periodicidad elegida por el usuario. Concretamente es una aplicación que permite proporcionar entradas a Cron, que es el programa que realmente ejecuta las tareas periódicamente. Cada usuario del sistema posee un Crontab personalizado, y solo puede hacer uso de la aplicación si su nombre aparece en /etc/cron.allow.

La sintaxis del archivo Crontab es la siguiente: hay seis capos por cada línea, cada uno separado por un espacio. Los cincos primeros campos especifican el momento preciso de ejecución y el sexto es el comando o script que se ejecutará.

1	**Minuto (0 al 59)**
2	Hora (0 al 23)
3	Dia (1 al 31)
4	Mes (1 al 12, o nombres)
5	Día de la semana (0 al 7, 0=7=Domingo, o nombres)
6	Comando a ejecutar a ejecutar

El valor '*' indica 'a cualquier [hora/dia/etc]'. Y el */TIEMPO indica 'ejecutar cada cierto TIEMPO'. Por ejemplo Para apagar el equipo a las siete y media de la mañana del 15 de febrero (ojo, hay que ser root para ejecutar esto):

```
30  7  15  2  *          /sbin/shutdown -h now
```

SCRIPT PARA LA ACTUALIZAR LAS TABLAS DE GENE ONTOLOGY

El siguiente script es el encargado de descargar las tablas de GO necesarias para la base de datos Proyecto Genome:

```
#!/bin/bash
cd BD

# Directorio raiz del entorno de desarrollo de java JSK 1.5up6
export JAVA_HOME=/home/simo01/programs/jdk1.5.0_06

# Directorio raiz servidor TOMCAT
export CATALINA_HOME=/home/simo01/programs/jakarta-tomcat-4.1.31

# Variable de entorno necesaria para que funcionen las consultas SOAP
export CLASSPATH=$CLASSPATH:/home/simo01/programs/soap-2_3_1/lib/soap.jar ...

# Variables de entorno necesarias para correr MYSQL
export MYSQL_UNIX_PORT=/home/simo01/tmp/mysqld-simo01.sock
export MYSQL_TCP_PORT=3307

# Borrar los archivos antiguos y descargar del ftp
rm go_daily-termdb-data

wget ftp://ftp.godatabase.org/godatabase/archive/latest-termdb/go_daily-
       termdb-data.gz
# Descomprimir el nuevo archivo
gzip -d go_daily-termdb-data.gz

# Se vuelcan los datos del nuevo archivo a la tabla
/home/simo01/programs/mysql-standard-5.0.18-linux-x86_64-glibc23/bin/mysql -
uroot --password=*** proyecto_genome < go_daily-termdb-data
```

SCRIPT PARA ACTUALIZAR LAS TABLAS DE ANOTACIONES GO

El siguiente script es el encargado de descargar las tablas de anotaciones de GO:

```
#!/bin/bash

cd BD
rm gene_association.*

# Directorio raiz del entorno de desarrollo de java JSK 1.5up6
export JAVA_HOME=/home/simo01/programs/jdk1.5.0_06

# Directorio raiz servidor TOMCAT
export CATALINA_HOME=/home/simo01/programs/jakarta-tomcat-4.1.31

# Variable de entorno necesaria para que funcionen las consultas SOAP
export CLASSPATH=$CLASSPATH:/home/simo01/programs/soap-2_3_1/lib/soap.jar ...

# Variables de entorno necesarias para correr MYSQL
export MYSQL_UNIX_PORT=/home/simo01/tmp/mysqld-simo01.sock
export MYSQL_TCP_PORT=3307


#DESCARGA DE TABLAS ANOTACIONES

wget http://www.geneontology.org/cgi-
bin/downloadGOGA.pl/gene_association.sgd.gz
gzip -d gene_association.sgd.gz

wget http://www.geneontology.org/cgi-
bin/downloadGOGA.pl/gene_association.mgi.gz
gzip -d gene_association.mgi.gz

wget http://www.geneontology.org/cgi-
bin/downloadGOGA.pl/gene_association.rgd.gz
gzip -d gene_association.rgd.gz

wget http://www.geneontology.org/cgi-
bin/downloadGOGA.pl/gene_association.cgd.gz
gzip -d gene_association.cgd.gz

wget http://www.geneontology.org/cgi-
bin/downloadGOGA.pl/gene_association.goa_human.gz
gzip -d gene_association.goa_human.gz
```

```
/home/simo01/programs/mysql-standard-5.0.18-linux-x86_64-glibc23/bin/mysql -
uroot --password=*** -e "use proyecto_genome; delete from go_association_sgd;
load data infile '/home/simo01/BD/gene_association.sgd' into table
go_association_sgd; delete from go_association_mgi; load data infile
'/home/simo01/BD/gene_association.mgi' into table go_association_mgi; delete
from go_association_rgd; load data infile
'/home/simo01/BD/gene_association.rgd' into table go_association_rgd; delete
from go_association_cgd; load data infile
'/home/simo01/BD/gene_association.cgd' into table go_association_cgd; delete
from go_association_hs; load data infile
'/home/simo01/BD/gene_association.goa_human' into table go_association_hs;"
```

SCRIPT PARA ACTUALIZAR LAS TABLAS DE ENTREZ GENE

El siguiente script es el encargado de descargar las tablas de Entrez Gene necesarias para la base de datos Proyecto Genome:

```
#!/bin/bash

cd BD
rm gene2go
rm gene2accession
rm gene2unigene

# Directorio raiz del entorno de desarrollo de java JSK 1.5up6
export JAVA_HOME=/home/simo01/programs/jdk1.5.0_06

# Directorio raiz servidor TOMCAT
export CATALINA_HOME=/home/simo01/programs/jakarta-tomcat-4.1.31

# Variable de entorno necesaria para que funcionen las consultas SOAP
export CLASSPATH=$CLASSPATH:/home/simo01/programs/soap-2_3_1/lib/soap.jar ...

# Variables de entorno necesarias para correr MYSQL
export MYSQL_UNIX_PORT=/home/simo01/tmp/mysqld-simo01.sock
export MYSQL_TCP_PORT=3307

#DESCARGA DE GENE2GO

wget ftp://ftp.ncbi.nlm.nih.gov/gene/DATA/gene2go.gz
gzip -d gene2go.gz
/home/simo01/programs/mysql-standard-5.0.18-linux-x86_64-glibc23/bin/mysql -
uroot --password=*** -e  "use proyecto_genome; delete from gene2go; load data
infile '/home/simo01/BD/gene2go' into table gene2go;"
```

```
#DESCARGA DE GENE2ACCESSION

wget ftp://ftp.ncbi.nlm.nih.gov/gene/DATA/gene2accession.gz
gzip -d gene2accession.gz
/home/simo01/programs/mysql-standard-5.0.18-linux-x86_64-glibc23/bin/mysql -
uroot --password=*** -e  "use proyecto_genome; delete from gene2accession;
load data infile '/home/simo01/BD/gene2accession' into table gene2accession;"

#DESCARGA DE GENE2UNIGENE

wget ftp://ftp.ncbi.nlm.nih.gov/gene/DATA/gene2unigene
/home/simo01/programs/mysql-standard-5.0.18-linux-x86_64-glibc23/bin/mysql -
uroot --password=*** -e  "use proyecto_genome; delete from gene2unigene; load
data infile '/home/simo01/BD/gene2unigene' into table gene2unigene;"
```

PERÍODO DE ACTUALIZACIÓN

Actualmente se actualizan todas las tablas de la base de datos cada día a las 0:00 horas. El fichero Crontab tiene por tanto el siguiente contenido:

```
#M h DOM MON DOW COMMAND

0  0  *  *  *     . ./BD/Scripts/Download_and_Load_Anotaciones.sh
0  0  *  *  *     . ./BD/Scripts/Download_and_Load_GO.sh
0  0  *  *  *     . ./BD/Scripts/Download_and_Load_EntrezGene.sh
```

4.2. Diseño e implementación de la aplicación

4.2.1 Implementación de las Clases y del web service

La estructura de nuestro proyecto (en cuanto a código) la hemos dividido en dos paquetes: el paquete “util” y el paquete “genome”. El primero de ellos lo hemos diseñado para albergar toda aquella clase referente a “utilities”, es decir, métodos que se van a utilizar en el resto de las clases, como por ejemplo, la inicialización de la conexión a la base de datos. El segundo paquete, “genome”, lo hemos establecido para realizar todo lo referente a SOAP y a la especificación de las querys.

A continuación vamos a ver más específicamente qué contiene cada clase y una explicación más detallada de sus métodos.

EL PAQUETE UTIL

Este paquete consta de las siguientes clases:

- Initial_Parameters.java
- Get_Text.java
- DBManager.java
- GenObject.java
- ParserGO.java
- Utilities.java

Initial_Parameters.java

Esta clase implementa el acceso al fichero "web.xml" del proyecto, donde residen datos importantes. Unos datos importantes a considerar y que pueden ser modificables, bien por traslado de la base de datos de un sistema de gestión a otro, o bien por traslado del proyecto de un servidor a otro, son: la ruta de nuestra base de datos, el login y password para acceder a mysql, y la url necesaria para SOAP.

De esta forma, si se produce algún cambio de login, password, traslado del proyecto de un servidor a otro, etc. Sólo habría que modificar estos parámetros en el fichero "web.xml" y no modificar nada del código del proyecto. Así obtenemos modularidad y fácil accesibilidad.

Dentro de la clase "Initial_Parameters.java", los parámetros anteriormente mencionados e incluidos en el "web.xml" son:

```
public String dbSource;
public String dbLogin;
public String dbPassword;
public String url;
```

Éstos son declarados como atributos en dicha clase. Ésta clase únicamente tiene como método el constructor (además de los métodos get y set), en el cuál se accede al fichero "web.xml" y se obtienen los datos. Para leer los datos del fichero "web.xml" hemos utilizado la librería "jdom", una librería que permite manejar ficheros xml con facilidad.

Teniendo en cuenta que nuestro proyecto siempre tiene que estar dentro del "webapps" de algún Tomcat, el fichero "web.xml" debe de estar en la siguiente ruta:

```
webapps/ProyectoGenome/WEB-INF/web.xml
```

Get_Text.java

La página web que hemos diseñado para nuestro proyecto puede ser leída en dos idiomas: inglés y español. Para que esto sea posible, hemos creado un fichero "text.xml" donde se guardan todas las traducciones. Por ejemplo, si en nuestra pagina web tenemos una frase como "Introduzca los siguientes parámetros", entonces, en el fichero "text.xml" habrá dos entradas para esa frase una en español y otra con su traducción en inglés. Veamos más claramente cómo está estructurado este fichero con un ejemplo de él:

```
<text>
    <sentence_step4>
        <spanish>4-Introduzca los parametros para la consulta </spanish>
        <english>4-Set the parameters for the query</english>
    </sentence_step4>

    <list-of-genes>
        <spanish>Lista de genes:</spanish>
        <english>List of genes:</english>
    </list-of-genes>
</text>
```

En este ejemplo podemos ver que la raíz de este fichero es <text>. Por lo tanto, toda frase o traducción debe ser hijo de esta raíz. Por ejemplo, vemos que <text> tiene un hijo <list-of-genes> que a su vez tiene dos hijos, uno que contiene la traducción en español (<spanish>) y otro contiene la traducción en inglés (<english>).

La clase "Get_Text.java" implementa el acceso a este fichero que está ubicado en:

`webapps/ProyectoGenome/WEB-INF/text.xml.`

Ésta clase sólo tiene un método cuya cabecera y explicación es la siguiente:

- **String get_text(String sentence, String language):** método que devuelve el significado de una frase, catalogada en el fichero "text.xml" con la entrada indicada por el primer parámetro (sentence) y el idioma indicado por el segundo parámetro (language).

DBManager.java

Ésta clase implementa la conexión a la base de datos, ejecución de una sentencia en la base de datos y finalización de la conexión. Como para acceder a la base de datos (crear conexión) se necesitan el usuario, login y ruta de la base de datos, es necesario llamar a la clase "Initial_Parameter.java" para obtener dichos parámetros.

Los métodos implementados por esta clase son:

- **void init_db():** inicializa la conexión con la base de datos. En este método se llama a la clase "Initial_Parameter.java" y se crea un objeto del tipo Connecion y otro del tipo Statement, necesarios para realizar las consultas a la base de datos.
- **void end_db():** método que finaliza la conexión a la base de datos.
- **ResultSet ejecutarQuery(String s1):** método que ejecuta una consulta a la base de datos y devuelve el resultado obtenido en un objeto del tipo ResultSet.
- **int ejecutarUpdate(String s1):** método que ejecuta un Insert, Delete o Update en la base de datos y devuelve si la acción ha producido algún error.

GenObject.java

Ésta clase se ha implementado para facilitar tanto la visualización como la obtención de resultados de una query. Como en la mayoría de las consultas, los resultados que se van a visualizar en la pagina web son los que en nuestra base de datos están etiquetados como "DB_Object_Symbol" y "GO_id", hemos diseñado esta clase para mayor comodidad. Así cuando de una consulta nos devuelvan un ResultSet, de éste ResultSet sólo vamos a coger las columnas que nos interesa visualizar, es decir, las etiquetadas como "DB_Object_Symbol" y "GO_id". Así cada iteración del ResultSet la vamos a transformar a un objeto GenObject. Por lo tanto, un objeto GenObject tendrá los parámetros:

```
String DB_Object_Symbol;

String GOid;
```

Ésta clase tiene implementados el constructor y los métodos get y set. Como constructores, tiene dos, uno sin parámetros y otro con parámetros. La cabecera del constructor con parámetros es:

- **GenObject(String object_Symbol, String oid):** crea un objeto GenObjecto y asigna a "DB_Object_Symbol" el primer parámetro, y a "GO_id" el segundo parámetro.

ParserGO.java

Para rellenar nuestra base de datos nos hacía falta un *parser* ya que las tablas que utilizamos estaban ya en otras bases de datos, por ejemplo la de GeneOntoloty. En www.geneontology.org se pueden obtener los ficheros .sql que contienen la estructura de cada tabla y los datos que deben almacenar.

La estructura de cada tabla se puede crear directamente desde mysql con la sentencia ('nombre_fichero.sql' es el fichero con la sentencia *create* para construir cada tabla):

```
source 'nombre_fichero.sql'
```

El relleno de las tablas se puede hacer con el parser que hemos creado. Este parser lee los datos de un determinado fichero (la ruta del fichero hay que modificarla dentro del código), y realiza un volcado de la información leída al interior de la tabla. Para realizar este volcado, necesitamos crear la conexión con la base de datos. Para ello utilizamos la clase "DBManager.java".

La clase "ParserGO.java" contiene el método main, desde donde se le dice el archivo que contiene los datos de relleno de la tabla, y también tiene otro método cuya cabecera es la siguiente:

- **void fill_DB(String[] vector):** rellena una fila de una tabla (indicada en el código) con los parámetros de entrada. Cada componente del vector de entrada se corresponde con cada columna de la tabla a rellenar.

Pero el volcado de la información también se puede realizar sin la necesidad del parser. La sentencia de mysql que realiza esta acción es:

```
Load data infile 'nombre del fichero.txt' into table 'nombre de la tabla';
```

Utilities.java

Esta clase contiene cuatro métodos que son utilizados por dos clases del paquete genome (que explicaremos a continuación). Los cuatros métodos y sus correspondientes cabeceras son:

- **String data_from_vector (Vector v, String where)**: este método genera la parte correspondiente del "where" de un "select", es decir, genera las condiciones de un select.
- **String data_from_resultset (ResultSet rs,String where,String column)**: este método realiza la misma funcionalidad que el método anterios. La única diferencia es que éste método coge los valores que necesita de un ResultSet, y el método anterior los recoge de un Vector.
- **boolean contains (Vector v,GenObject go)**: este método devuelve verdadero si en el vector de GenObject que se le pasa como primer parámetro está el GenObject que se le pasa como segundo parámetro.
- **Vector edit_results (Vector v_genobjects)**: este método formatea el resultado de la query. Es decir, el resultado final obtenido por una query es pasado a un vector de GenObject (como ya se ha explicado anteriormente), y éste método genera los dos String que se van a mostrar por pantalla a partir del vector de GenObjects.

EL PAQUETE GENOME

Este paquete está compuesto por cuatro clases. Tres de ellas son las que se encargan de implementar el cliente y servidor del web service.

Las clases de este paquete son:

`CargarCombos.java`

`AccessDataBaseClient.java`

`AccessDataBaseServiceGO.java`

`AccessDataBaseServiceEntrez.java`

Las tres últimas clases son las encargadas de realizar la conexión y comunicación mediante SOAP. Cuando en la pagina web se está en el último paso de cada query se llama a un método de la clase "AccessDataBaseClient.java", que es el encargado de crear la conexión mediante una url y una uri. Una vez creada la conexión se envían los parámetros al método concreto de la clase "AccessDataBaseServiceGO.java", si estamos ejecutando querys relativas a la base de datos de Gene Ontology, o de la clase "AccessDataBaseServiceEntrez.java", si estamos ejecutando querys relatiavas a la base de datos de Entrez Gene, y se ejecuta la query accendiendo a nuestra base de datos. Una vez obtenido el resultado es recibido por la clase "AccessDataBaseClient.java", la cual lo devuelve a la pagina web para que muestre el resultado. El proceso SOAP descrito anteriormente se explica más detalladamente cuando hablemos de la implementación de la clase "AccessDataBaseClient.java".

CargarCombos.java

Ésta clase es llamada desde cada jsp cuando se necesita acceder a la base de datos para obtener un conjunto de datos, por ejemplo los GO_id correspondientes al organismo "taxon:4932". Éstos datos serán utilizados para rellenar los datos que se visualizarán en cada ComboBox de nuestra página web.

Ésta clase sólo tiene un método cuya cabecera es la siguiente:

- **ResultSet cargaCombo (String query)**: éste método simplemente inicializa una conexión con la base de datos, ejecuta la consulta pasada por parámetro y cierra la conexión con la base de datos.

AccessDataBaseClient.java

Esta clase es esencial para el web service, ya que es la que implementa verdaderamente la utilización del web service. Como hemos explicado anteriormente, cuando se llega al último paso de ejecutar una consulta en nuestra página web (después de elegir el organismo, los genes…) se llama a un método de esta clase (get_result() o get_result_gene2go() dependiendo de a qué base de datos se refiera la consulta). Éste

método llama a su vez a otro método (getData()) que es el encargado crear el método Call, decir dónde está definido el método SOAP, invocar la url correcta y llamar al método correspondiente de la clase "AccessDataBaseService.java" para que devuelva el valor de la query.

Antes de nada, necesitamos crear un fichero xml donde se definan los métodos que vamos a llamar mediante SOAP y los parámetros que necesitamos mandarle. Los métodos que podemos invocar mediante SOAP son los implementados en "AccessDataBaseServiceGO.java" y "AccessDataBaseServiceEntrez.java", que son exáctamente 8 (4 consultas por cada base de datos utilizada(GeneOntology, EntrezGene). Todos los métodos tienen los mismos parámetros de entrada, aunque su función sea distinta en cada método. El fichero xml será utilizado por el web service para saber qué tiene que recibir cada método definido. El contenido de este fichero (llamado "AccessDataBaseServiceGo.xml") para describir los métodos de "AccessDataBaseServiceGO.java"es:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment" id="urn:databaseGO"
checkMustUnderstands="false">
     <isd:provider type="java" scope="Request" methods="getQuery1 getQuery2 getQuery3
    getQuery4">
      <isd:java class="genome.AccessDataBaseServiceGO"static="false" />
     </isd:provider>
     </isd:service>
```

Y el contenido del otro fichero para describir los métodos de "AccessDataBaseServiceEntrez.java", llamado "AccessDataBaseServiceEntrez.xml" es:

```
<isd:service xmlns:isd=http://xml.apache.org/xml-soap/deployment
id="urn:databaseEntrez" checkMustUnderstands="false">
<isd:provider type="java" scope="Request" methods="getQuery1_EntrezGene
getQuery2_EntrezGene getQuery3_EntrezGene getQuery4_EntrezGene">
 <isd:java class="genome.AccessDataBaseServiceEntrez" static="false" />
    </isd:provider>
</isd:service>
```

El contenido de estos fichero debe ser conocido por el web service. Para ello ejecutamos el siguiente código (en Linux puesto que nuestro proyecto está en un servidor bajo Linux (servidor marbore)):

```
echo "generando urn:database en SOAP"
echo ""
    java org.apache.soap.server.ServiceManagerClient
    http://marbore.dacya.ucm.es:2006/soap/servlet/rpcrouter deploy WEB-
    INF/src/genome/AccessDataBaseServiceGO.xml
java org.apache.soap.server.ServiceManagerClient
http://marbore.dacya.ucm.es:2006/soap/servlet/rpcrouter deploy WEB-
INF/src/genome/AccessDataBaseServiceEntrez.xml

echo ""
```

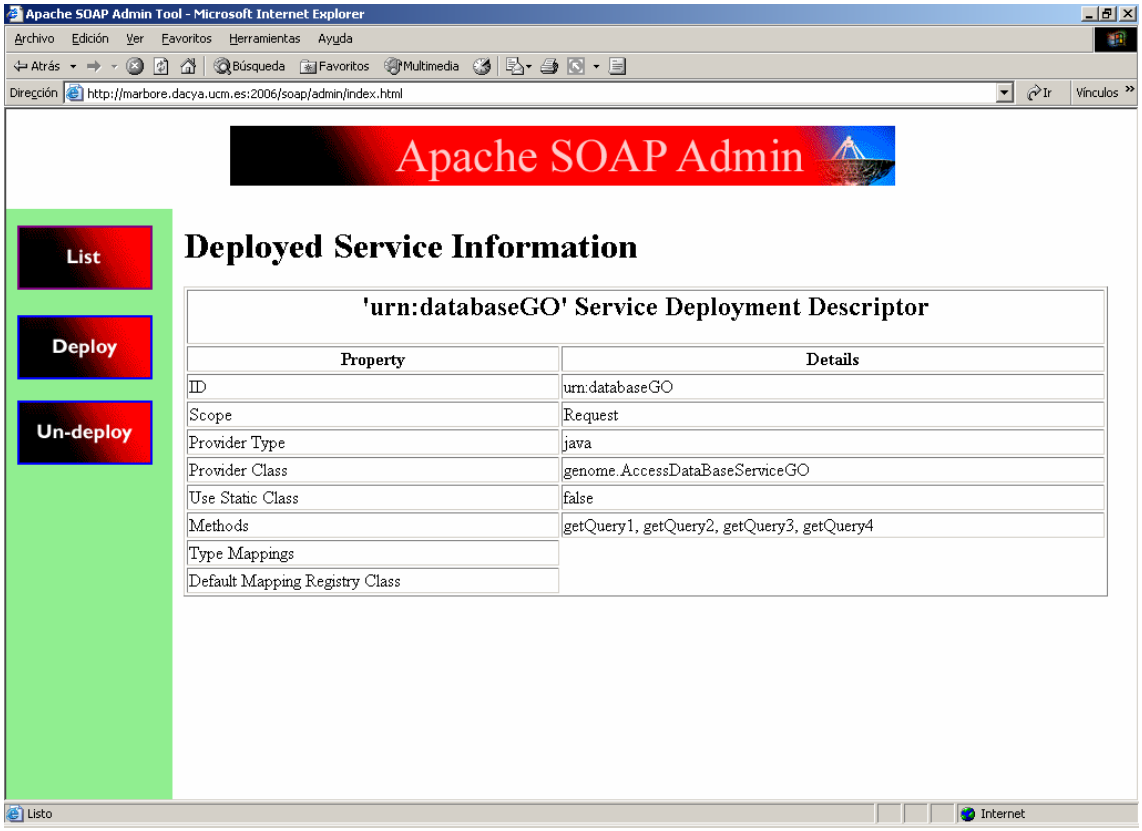
Una vez hecho esto ya sabemos que nuestro web service conoce la existencia de estos métodos. Para mayor seguridad podemos ejecutar el administrador del cliente de Apache SOAP, escribiéndo en la url de nuestro navegador lo siguiente:

http://localhost:8080/soap/admin/index.html

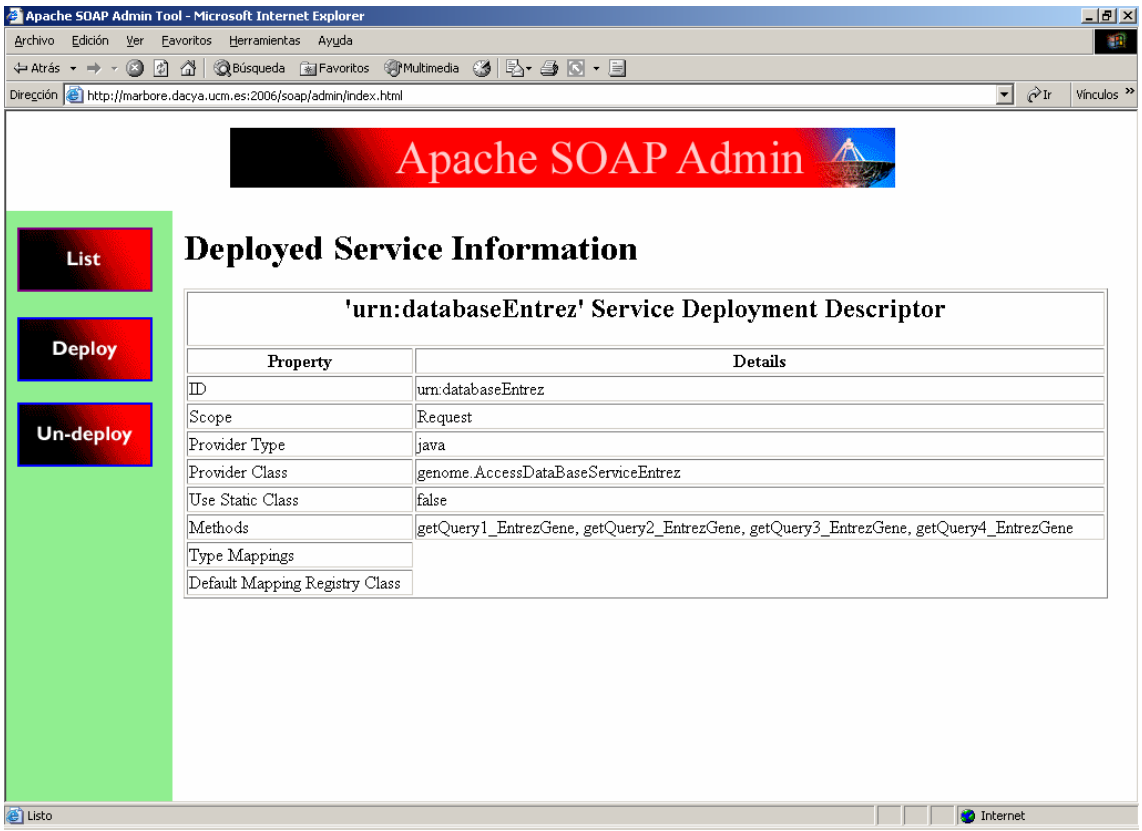
(teniendo en cuenta que localhost lo sustituiremos por la dirección del servidor y 8080 es el puerto en el que corre Apache Tomcat)

Al escribir esa url nos sale la pantalla inicial del administrador de cliente de Apache SOAP y le damos a 'Run'. Esto nos lleva a otra pantalla donde podemos realizar varias acciones. Lo que queremos hacer es listar la información de las URIs donde hemos especificado nuestros métodos (la hemos llamado urn:databaseGO y urn:databaseEntrez). Por lo tanto, en la pantalla actual le damos al botón List y pulsamos sobre los dos links y obtenemos las siguientes pantallas:

urn:databaseGO



urn:databaseEntrez



Una vez descrito el proceso para especificar la información que vamos a utilizar en el web service mediante SOAP, sigamos explicando la clase "AccessDataBaseClient.java". Ahora nombramos los métodos que implementa y explicaremos los más importantes:

- **Vector get_result(String gen, String ont, String slims, String consult, String table_of):** si la base de datos de la consulta es GeneOntology, llamamos a este método para que a su vez llame al método encargado de realizar la funcionalidad de SOAP.
- **Vector get_result_gene2go(String gen, String slims, String consult, String taxon):** si la base de datos de la consulta es EntrezGene, llamamos a este método para que a su vez llame al método encargado de realizar la funcionalidad de SOAP.
- **Vector getData (Vector gen, String ont, Vector slims, String method, String table_of, String db) throws SOAPException, MalformedURLException**: este método es el que engloba toda la funcionalidad del web service y SOAP. Vamos a ver parte del código que implementa esta función:

```
//call to SOAP
Call call=new Call();
if (db.equals("GO"))
    //urn:database is where is defined the method SOAP
    call.setTargetObjectURI("urn:databaseGO");
else
    //urn:database is where is defined the method SOAP
    call.setTargetObjectURI("urn:databaseEntrez");

//set method name to call
call.setMethodName(method);
call.setEncodingStyleURI("http://schemas.xmlsoap.org/soap/encoding/");
```

Aquí podemos observar que creamos un objeto Call que será el encargado de invocar el método correspondiente de "AccessDataBaseServiceGO.java" o de "AccessDataBaseServiceEntrez.java". Después de crear el objeto Call, ponemos la URI que hemos definido previamente en el fichero "AccessDataBaseServiceGO.xml" o "AccessDataBaseServiceEntrez.xml". A continuación le pasamos el método del nombre al que queremos llamar (método de "AccessDataBaseServiceGO.java" o de "AccessDataBaseServiceEntrez.java") y ponemos a la URI donde se define el tipo de encriptación utilizada para los mensajes SOAP.

Una vez hecho esto, creamos los parámetros que le tenemos que pasar al método y los metemos en un vector en el orden apropiado, es decir, en el la posición 0 del vector introducimos el primer parámetro, en la posición 1 el segundo, y así sucesivamente.

```
Vector paramList=new Vector();
paramList.addElement(param1);
paramList.addElement(param2);
paramList.addElement(param3);
paramList.addElement(param4);
call.setParams(paramList);
```

Después ponemos la URL del rpcrouter de SOAP, recordemos que esta URL la tenemos almacenada en nuestro "web.xml", por lo tanto tendremos que acceder a él para obtener este dato:

```
Response resp=call.invoke(url,"");
```

Una vez hecho esto, el web service envía un mensaje SOAP con todos los parámetros hacia la clase "AccessDataBaseServiceGO.java" o "AccessDataBaseServiceEntrez.java" (dependiendo de la URI definida) y más exactamente al método del cual queremos obtener el resultado de la query. Una vez que el método ejecuta sus instrucciones, devuelve el resultado que es recogido por el método que estamos definiendo en estos momentos mediante:

```
Parameter result=resp.getReturnValue();
```

Así hemos realizado el envío y recepción de un mensaje SOAP en un web service.

- **Vector get_genes(String gen):** en la página web se introducen ciertos parámetros (genes, go ids…) como una lista cuyos componentes están separados por espacios. Éste método es el encargado de extraer cada componente e introducirlo en un Vector.

AccessDataBaseServiceGO.java

Esta clase es la que recibe los mensajes SOAP referentes a consultas de la base de datos de Gene Ontology mandados por la clase "AccessDataBaseClient.java". Consta de 4 métodos y todos ellos tienen la misma estructura:

1. Iniciar la conexión con la base de datos.
2. Ejecutar la consulta (distinta en cada método)
3. Cerrar la conexión con la base de datos.
4. Elaborar el resultado como un Vector de GeneObject.

Los parámetros que envía el método SOAP siempre son 4 para las consultas de las bases de datos 'GeneOntoloty'. Cada parámetro representa cosas distintas teniendo en cuenta la base de datos a tratar en cada caso.

La cabecera general de los métodos es:

```
Vector getQuery(Vector gen, Vector slims, String ont, String table_of);
```

y el significado de los parámetros es:

- **gen**: es un vector de DB_Object_Symbol que el usuario elige dentro de un ComboBox en la página web.
- **slims**: sólo útil para las consultas 3 y 4. Representa un conjunto de identificadores GO (GO_id) elegidos por el usuario en un ComboBox.
- **ont**: es el "aspect" que queremos (biological process, cellular component o molecular function).
- **table_of**: representa la tabla que estamos tratando (go_association_sgd, go_association_cgd…)

Las cabeceras de los 4 métodos son:

- **Vector getQuery1(Vector gen, Vector slims, String ont, String table_of)**: ejecuta la consulta nº 1 para la base de datos GeneOntology.
- **Vector getQuery2(Vector gen, Vector slims, String ont, String table_of)**: ejecuta la consulta nº 2 para la base de datos GeneOntology.
- **Vector getQuery3(Vector gen, Vector slims, String ont, String table_of)**: ejecuta la consulta nº 3 para la base de datos GeneOntology.
- **Vector getQuery4(Vector genes, Vector slims, String ont, String table_of)**: ejecuta la consulta nº 4 para la base de datos GeneOntology.

AccessDataBaseServiceEntrez.java

Esta clase es la que recibe los mensajes SOAP referentes a consultas de la base de datos de Entrez Gene mandados por la clase "AccessDataBaseClient.java". Consta de 4 métodos y todos ellos tienen la misma estructura (estructura igual a los métodos de su clase hermana "AccessDataBaseServiceEntrez.java".

Si la base de datos a tratar es '**EntrezGene'**, la cabecera general de los métodos es:

```
Vector getQuery_EntrezGene(Vector gen, Vector slims, String ont, String taxon);
```

y el significado de los parámetros es:

- **gen**: es un vector de 'gene_id' que el usuario elige dentro de un ComboBox en la página web.
- **slims**: sólo útil para las consultas 3 y 4. Representa un conjunto de identificadores GO (go_id) elegidos por el usuario en un ComboBox.
- **ont**: es el "aspect" que queremos (biological process, cellular component o molecular function).
- **taxon**: representa el organismo que estamos tratando.

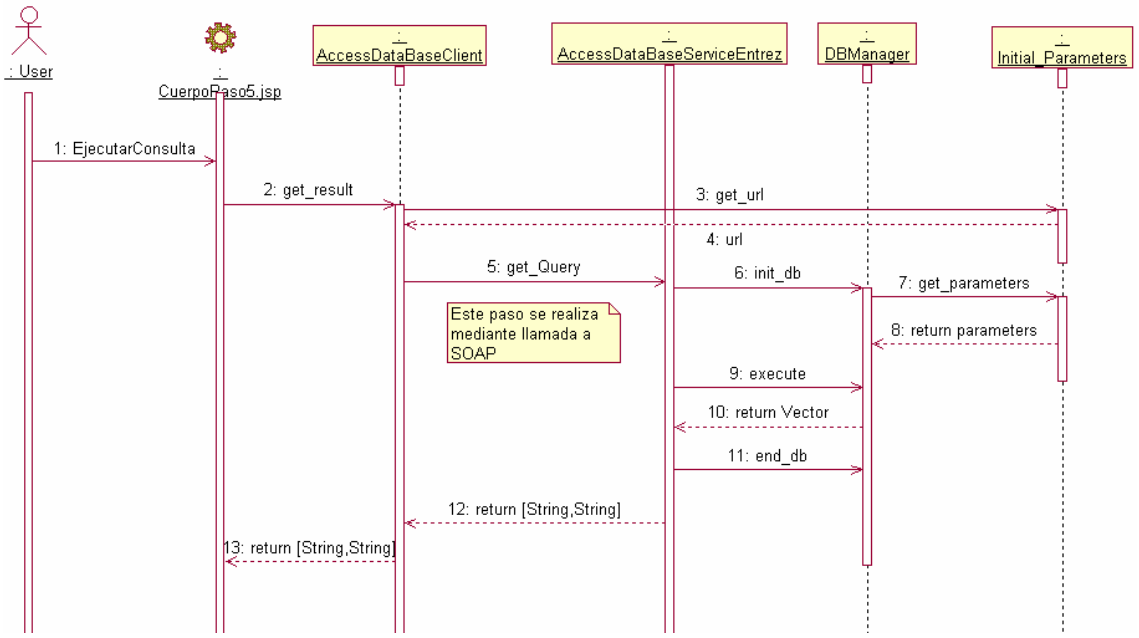
Las cabeceras de los 4 métodos son:

- **Vector getQuery1_EntrezGene(Vector gen, Vector slims, String taxon)**: ejecuta la consulta nº 1 para la base de datos EntrezGene (tabla gene2go).
- **Vector getQuery2_EntrezGene(Vector gen, Vector slims, String taxon)**: ejecuta la consulta nº 2 para la base de datos EntrezGene (tabla gene2go).
- **Vector getQuery3_EntrezGene(Vector gen, Vector slims, String taxon)**: ejecuta la consulta nº 3 para la base de datos EntrezGene (tabla gene2go).
- **Vector getQuery4_EntrezGene(Vector gen, Vector slims, String taxon)**: ejecuta la consulta nº 4 para la base de datos EntrezGene (tabla gene2go).
- **Vector look_Other(Vector v_genobjects, String genes, String ont, String taxon, String colum, boolean b, Vector slims)**: ejecuta cada consulta pedida para la tabla gene2accession.
- **Vector look_Other_Unigene(Vector v_genobjects, String genes, String ont, String taxon, boolean b, Vector slims)**: ejecuta cada consulta pedida para la tabla gene2unigene.

Diagrama de secuencia

Cuando el usuario realiza una consulta en nuestro sistema, se realizan una serie de llamadas entre clases, que vamos a ver en el diagrama de secuencia.

Pongamos por caso que nuestro usuario quiere realizar una consulta de la base de datos EntrezGene, e introduce los nombres de genes a consultar y todos los parámetros indicados en cada paso. Pues bien, cuando llega al último paso se ejecuta la consulta (paso 1). Esto hace que nuestro jsp 'CuerpoPaso5' llame al método 'get_result'



de la clase 'AccessDataBaseClient' (paso 2), pasándole obviamente todos los parámetros que el usuario haya introducido hasta el momento.

Después, la clase 'AccessDataBaseClient' es la encargada de llamar al método SOAP, previamente tiene que saber su url y para ello consulta ese parámetro llamando a la clase 'Initial_Parameters' (pasos 3 y 4).

Una vez que la clase 'AccessDataBaseClient' sabe la url con la que tiene que llamar al método SOAP, se produce la invocación de SOAP. Mediante SOAP se llama al método apropiado (getQuery1_EntrezGene, getQuery1_EntrezGene, getQuery1_EntrezGene o getQuery1_EntrezGene, dependiendo de la consulta que haya pedido el usuario) de la clase 'AccessDataBaseServiceEntrez'(paso 5).

Para realizar la consulta necesaria, ésta clase tiene que saber el nombre de la base de datos a consultar en MySQL, el nombre de usuario y la base de datos, para poder iniciar

la conexión con la base de datos (paso 6). Para conocer todos estos datos realiza una llamada a la clase 'Inicial_Parameters' (pasos 7 y 8).

Una vez obtenidos estos datos, se realiza la consulta (paso9 y 10) y se cierra debidamente la conexión con la bases de datos (paso 11).

Una vez hechos estos pasos, la clase 'AccessDataBaseServiceEntrez' ya ha obtenido los resultados requeridos, es decir, el resultado de la consulta que había hecho el usuario. Puesto que el resultado se requiere en dos formatos distintos, se devuelve un vector con dos String, uno con cada formato del resultado. Éste resultado es devuelto por SOAP a la clase que lo invocó, la clase 'AccessDataBaseClient' (paso 12).

Y finalmente, el resultado es devuelto al jsp que provocó la primera llamada, el jsp 'CuerpoPaso5'.

El diagrama de casos de uso es similar cuando se ejecuta una consulta a la base de datos de GeneOntology, sólo cambiarían dos cosas:

- el jsp 'CuerpoPaso5' no llama al método 'get_result', sino al método 'get_result_gene2go'.
- La clase 'AccessDataBaseClient' no llama a la clase 'AccessDataBaseServiceEntrez', sino a la clase 'AccessDataBaseServiceGO'.

4.3. Diseño de la Interfaz Web

En este apartado pretendemos dar una visión directa del uso de las tecnologías tratadas en el apartado 3.2 para el caso concreto de nuestro proyecto, es decir, que estructura HTML hemos seguido, que fragmentos de código incrustado con etiquetas de JSP se han empleado, como hemos aplicado las hojas de estilo (CSS) sobre los archivos .jsp que implementan la estructura web, que funciones JavaScript nos han ayudado a conseguirlo, etc. Para ello vayamos por partes:

4.3.1 HTML en el Proyecto Genome

Puesto que en nuestro proyecto no encontramos ningún documento HTML, hablaremos aquí únicamente de la estructura de una página típica (CuerpoPaso2.jsp) dejando a un lado las secciones de código incrustadas.

```
<html>
Comienzo del documento HTML.
<head>
   <title>UCM. Proyecto Sistemas Informaticos.</title>
   <meta http-equiv="Content" content="text/html; charset=iso-8859-1">
    <link rel="stylesheet" href="css/genomeStyle1.css" type="text/css"    id="estilo">
</head>
```

Como ya dijimos, en la cabecera ponemos información del documento, título, tipo de contenido y la hoja de estilos de la que toma los parámetros.

```
<body background="images/fondo.jpg">
```

Comienzo del cuerpo del documento, especificamos en ella la imagen que servirá de fondo.

```
<FORM name="elegirConsultaForm" method="GET" action="./CuerpoPaso3_1.jsp"
enctype="text/plain">
    <input Type="hidden"  name="db" value="GO">
    <button type=submit style="background-color:transparent;border:0px">
    <img src="images/ico_ok.gif" alt="OK" width="40" height="40"/>
    <FONT class="subtitulo">Gene Ontology </FONT>
    </button>
</FORM>
```

A continuación se encuentra el formulario 'elegirConsultaForm' que recibe datos del usuario donde selecciona la base de datos deseada, en este caso es GO, por lo que guarda en la variable db el valor 'GO' para realizar en pasos sucesivos la consulta sobre la base de datos.

Los demás puntos indican el icono donde hay que dar para aceptar y el estilo que se aplicará sobre el texto Gene Ontology.

Por ultimo:

```
<p><a href="javascript:history.go(-1);"><img src="images/ico_atras.gif"</a></p>

</body>

</html>
```

Ponemos un icono 'ico_atras' que es un link que al pulsar nos llevará a la pagina anterior almacenada en el historial del navegador, para obtener esta, utilizamos la función JavaScript 'history.go'.

4.3.2 JavaScript en el Proyecto Genome

Como ya dijimos en la descripción teórica del lenguaje JavaScript, debemos definir las funciones en cada uno de los documentos HTML (JSP, en nuestro caso) en que las vamos a usar. Además, no en un lugar cualquiera, para garantizar la accesibilidad, lo haremos en la sección <HEAD> del documento.

FUNCIONES USADAS Y BREVE DESCRIPCCIÓN:

```
<a href="javascript:history.go(-1);"> <img src="images/ico_atras.gif"…….
```

con esta instrucción, utilizada en la mayoría de nuestras páginas, aplicada a la imagen "Volver atrás", conseguimos que cuando el usuario quiere volver a la página anterior visitada (paso anterior en la secuencia de consulta) pueda hacerlo tan solo pulsando en la imagen. Al hacer esto, se desencadena la ejecución de un javascript que enlaza el navegador, con la página que se encuentra en la posición –1 del historial, es decir, la página anterior.

```
<SCRIPT LANGUAGE="JavaScript">

    function agregarGenALista()
    {
     lista = document.LayoutForm.ListaGenes.value;
     select = document.LayoutForm.Genes.value;
     document.LayoutForm.ListaGenes.value = lista + select + ' ';
    }
```

```
function agregarIdALista()
{
 lista = document.LayoutForm.ListaIds.value;
 select = document.LayoutForm.IdsGO.value;
 document.LayoutForm.ListaIds.value = lista + select + ' ';
}

</SCRIPT>
```

Estas dos funciones, muy similares en su estructura y funcionamiento, hacen que el usuario, una vez seleccionado un gen (o un identificador), dándole a añadir, se adjunte a la lista de genes (o lista de identificadores), es decir, el textarea que se encuentra inmediatamente debajo.

```
function actualizarListaGOSlim()
```

Esta función, dependiendo del tipo de GO slim predeterminado (en la web de Gene Ontology) y de la ontología (Biological Process, Molecular Function ó Cellular Component) muestra los términos GO correspondientes.

```
function ventanaPopUp (URL)
```

Esta función abre en una ventana nueva los resultados de la consulta para que el usuario pueda trabajar con ellos con mayor comodidad. Por ejemplo, para seleccionarlos y llevárselos a otro programa, o para guardarlos en un archivo.

VENTAJAS E INCONVENIENTES DE JAVASCRIPT APRECIADAS

Ventajas:

- Sencillez
- Muchos contenidos en internet.
- Rapidez al ser interpretado en el lado del cliente.
- El lenguaje de scripting es seguro y fiable porque está en claro y hay que interpretarlo, por lo que puede ser filtrado; para el mismo Javascript, la seguridad es casi total y sólo en su primera versión el CIAC (Computer Incident Advisory Committee) señaló problemas de leve entidad, entre ellos la lectura de la caché y de los sitios visitados, de la dirección e-mail y de los archivos presentes en el disco. Sin embargo, estos fallos se corrigieron ya en las versiones de Netscape sucesivas a la 2.0.

- El código Javascript se ejecuta en el cliente por lo que el servidor no es solicitado más de lo debido. Un script ejecutado en el servidor, sin embargo, sometería a éste a dura prueba y los servidores de capacidades más limitadas se podrían resentir de una continua solicitud por un mayor número de usuarios.

Desventajas:

- Redefinición en cada documento
- Los script tienen capacidades limitadas, por razones de seguridad, por lo cual no es posible hacer todo con Javascript, sino que es necesario usarlo conjuntamente con otros lenguajes evolucionados, posiblemente más seguros, como Java. Dicha limitación es aún más evidente si queremos operar en el hardware del ordenador, como, por ejemplo, la fijación en automático de la resolución vídeo o la impresión de un documento.
- Un problema importante es que el código es visible y puede ser leído por cualquiera, incluso si está protegido con las leyes del copyright.
- El código del script debe descargarse completamente antes de poderse ejecutar y ésta es la otra cara de la moneda de lo que hemos dicho anteriormente: si los datos que un script utiliza son muchos (por ejemplo, una recopilación de citas que se mostrara de manera casual), el tiempo que tardará en descargarse será muy largo, mientras que la interrogación de la misma base de datos en el servidor sería más rápida.

4.3.3 Java Server Pages (JSP) en el Proyecto Genome.

Ya hablamos anteriormente de la importancia y múltiples ventajas de utilizar JSP, ahora vamos a ver como hemos aplicado esta tecnología a nuestro proyecto. Para clarificar esto, mostraremos diferentes segmentos de código, indicaremos en que fichero se encuentran y comentaremos cual es su función.

En **index.jsp** encontramos:

```
<%@ page import="utils.*"%>
```

Esto será habitual a lo largo de los diferentes documentos, y se refiere a la importación de librerías.

(1)

```
if (session.getAttribute("language")==null)
        session.setAttribute("language","spanish");

if ((request.getParameter("change_language")!=null) &&
(request.getParameter("change_language").equals("spanish")))

session.setAttribute("language","spanish");

if ((request.getParameter("change_language")!=null) &&
(request.getParameter("change_language").equals("english")))

        session.setAttribute("language","english");
```

(2)

```
String name_step=(String) session.getAttribute("name_step");

if (name_step==null)
        name_step="CuerpoPaso1.jsp";
```

En este código, se presentan lo que pueden ser las acciones menos intuitivas del código JSP incrustado.

En el primer (1) punto empleamos la variable de sesión “language” en la que se guarda si el idioma actual de navegación es “spanish” ó “english”, así a lo largo del código veremos como consultamos si estamos en un idioma o en otro y, actuando en consecuencia, se pide la frase adecuada al archivo “text.xml” que contiene todo el texto en ambos idiomas.

Un análisis más minucioso del código nos revela que, si no hay un lenguaje seleccionado explícitamente, se mostrará la información en español, es decir, se asignara el valor “spanish” a la variable “language”.

Por otro lado, si se ha seleccionado la función de cambiar lenguaje, en nuestro caso pinchando sobre la bandera correspondiente, se analiza sobre cual ha sido y se establece el nuevo valor de la variable “language”.

En segundo (2) lugar, se declara una variable “name_step” que, como su nombre indica, guarda el nombre del paso de la navegación en que se encuentra el usuario, en este caso, estará visualizando en el marco principal de su navegador, la página “CuerpoPaso1.jsp”. Esta variable nos será útil cuando queramos actualizar el contenido de todos los marcos (los 4) simultáneamente. (Ej. Cuando el usuario quiera cambiar el idioma, cambiará en todos los marcos mostrados en el navegador).

Como en el caso de “language”, asignamos “CuerpoPaso1.jsp” como valor por defecto a “name_sep”.

En **CuerpoPaso1.jsp**

En este archivo, además de la anterior función para establecer “name_step” con el valor correspondiente, encontramos este código:

```
session.removeAttribute("consulta");
session.removeAttribute("db");
session.removeAttribute("table_of");
session.removeAttribute("taxon");
session.removeAttribute("Ontologia");
session.removeAttribute("ListaGenes");
session.removeAttribute("ListaIds");
```

De nuevo es bastante intuitiva su utilidad y se encarga de borrar todos los atributos que, paso a paso, el usuario fue introduciendo en su consulta anterior, así se evitarán posteriores problemas con valores desfasados.

En **CuerpoPaso2.jsp:**

```
if (request.getParameter("consulta")!=null)
session.setAttribute("consulta",request.getParameter("consulta"));
```

Este código captura el valor de la variable “consulta” (del formulario de elección de consulta del usuario en el primer paso de la web) y establece el atributo “consulta” con ese valor, de modo que en pasos posteriores será visible para las consultas a la base de datos y la selección de datos que deberán pedirse al usuario.

En sucesivos archivos se irán estableciendo todos los valores que hemos borrado en el paso anterior con idéntica estructura de código por lo que no los trataremos individualmente.

```
String l=(String) session.getAttribute("language");
String sentence2 =Get_Text.get_text("sentence2",l);

<FONT>class="styleEnunciado"><%=sentence2%></FONT>
```

Este es un código que también se encuentra a menudo en nuestro proyecto, debido a la presencia de texto en dos idiomas. En primer lugar, leemos la variable “language” y almacenamos su contenido (english ó spanish) en “l”, con este valor consultamos el archivo “text.xml” mediante la función apropiada, recibiendo en “sentence2” la cadena esperada y en el idioma indicado.

Por último, imprimimos en pantalla el contenido de “sentence2” con el formato que se establece en las hojas de estilo como “styleEnunciado”.

En **CuerpoPaso4_1.jsp:**

```
String query = "SELECT DISTINCT DB_Object_Symbol FROM "+table_of+" WHERE
DB_Object_Symbol IS NOT NULL ORDER BY DB_Object_Symbol ASC";
ResultSet rs= genome.cargarCombos.cargaCombo(query);
...
```

No vamos a detenernos a hacer un análisis en profundidad del significado de estas líneas, baste decir que, una vez recogidos todos los datos necesarios del usuario vía web, se realiza la consulta sobre la base de datos elegida para, en un paso posterior, mostrar o guardar los resultados.

```
<% if (!consulta.equals("4")){%>
codigo HTML1
…
<% if (!consulta.equals("3")){%>
codigo HTML2
…
```

También utilizamos código JSP, para seleccionar si se ejecutan unas u otras partes del código HTML, por ejemplo, si el usuario decide hacer un tipo de consulta, tal vez necesite introducir identificadores de genes, mientras que en otra consulta, lo que necesitará será escribir los nombres de los genes.

En el ejemplo, si el usuario, seleccionó una consulta de tipo 3, se ejecutará el código HTML2, sin embargo, si seleccionó una consulta de tipo 4 será su correspondiente código en que visualizará el navegador.

4.3.4 Cascade Style Sheets (CSS) en el Proyecto Genome

Para ilustrar el uso de las propiedades de la tecnología por CSS, veamos, en primer lugar, una definición genérica de estilo aplicada al selector BODY:

```
BODY {
        color: #FFFFFF;
        background-image: transparent;
        font-family: "Arial", sans-serif;
        font-size: 9pt;
        margin: 0px 0px 0px 0px;
        padding: 0px 0px 0px 0px;
        scrollbar-face-color: #AA384A;
        scrollbar-highlight-color: #777777;
        scrollbar-3dlight-color: #000000;
        scrollbar-darkshadow-color: #000000;
        scrollbar-shadow-color: #000000;
        scrollbar-arrow-color: #000000;
        scrollbar-track-color: #2D5392;
}
```

Aunque la definición es bastante intuitiva, observemos línea a línea:

1. Indicamos que el color predeterminado de la fuente sea #FFFFFF (blanco), es decir, que sus componentes RGB se valoren a FF.
2. La imagen de fondo será transparente.
3. El tipo de letra será Arial,
4. con un tamaño de 9 puntos.
5. Sin forzar márgenes adicionales.
6. y con los colores indicados para las barras de desplazamiento que aparecen cuando la pantalla no puede mostrarse en su totalidad (scrollbars).

Como ya hicimos mención en el desarrollo teórico, también podemos definir clases personalizadas, por ejemplo, para los títulos.

```
.styleEnunciado
{
        text-align: center;
        vertical-align: middle;
        color: #FFFFFF;
        font-size: 11pt;
        font-family: "Arial Black";
        font-weight: bold;
}
```

1. El nombre de la clase es 'styleEnunciado', por tanto, para aplicárselo a un texto, añadiremos <FONT **class="styleEnunciado"**> al tag FONT.
2. Alineación horizontal del texto centrada.
3. Alineación vertical del texto centrado.
4. Color de fuente, #FFFFFF (blanco).
5. Tamaño de Fuente, 11 puntos.
6. Tipo de letra, "Arial Black".
7. Grosor de la fuente aumentado (negrita).

5. Manual de usuario

Con el fin de que un usuario no encuentre ninguna dificultad en el manejo de nuestra interfaz, reseñamos aquí la forma genérica de realizar cada una de las consultas implementadas en la plataforma. Comencemos por el principio:

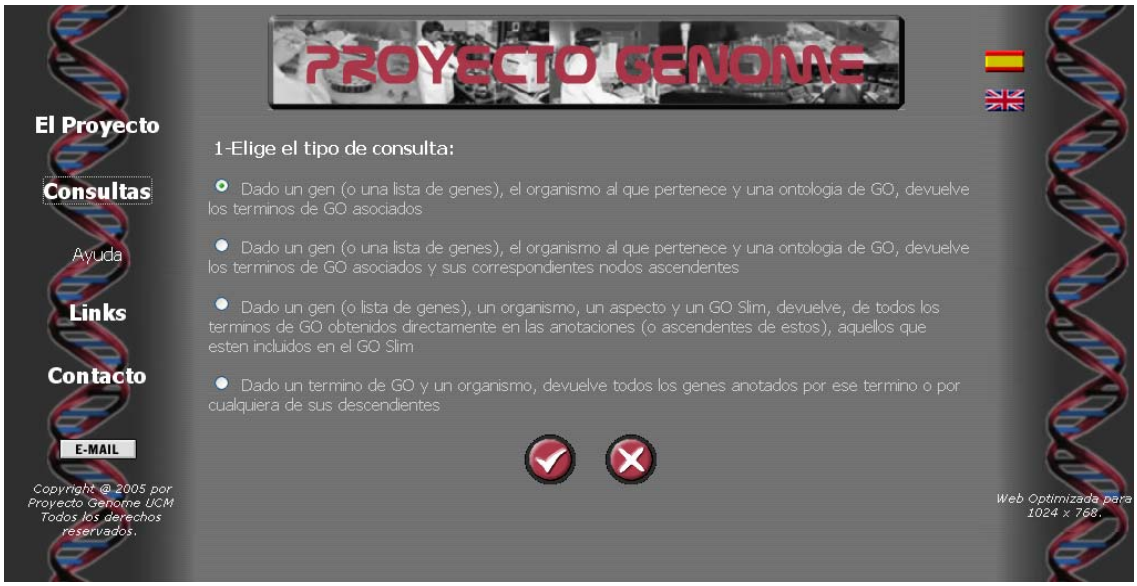
5.1 La Página de Inicio

Una vez en la web del Proyecto Genome, se encontrará la página de entrada al servicio, en la que podremos seleccionar directamente el tipo de consulta a realizar. Se encuentra estructurada en diversas partes:

- El menú de la izquierda, que se mantiene 'fijo' durante toda la consulta está compuesto por el esquema de navegación general para acceder a todas las opciones:
    - ***El proyecto***, breve explicación del contenido del proyecto.
    - ***Consultas***, el punto de partida para realizar cualquier consulta con nuestra herramienta.
    - ***Ayuda***, una referencia on-line a este manual.
    - ***Links***, enlaces a diversas páginas relacionadas con el proyecto.
    - ***Contacto***, la manera de ponerse en contacto con los desarrolladores del proyecto.

- El menú de la derecha, contiene los iconos para cambiar de idioma.

- Parte central de la pantalla, es la parte principal de la aplicación y la que irá cambiando en función de la opción elegida en el menú de la izquierda o el paso en el que nos encontremos en una consulta.



5.2 Las Consultas

A continuación vamos a explicar el proceso para realizar cada una de las cuatro consultas que hemos definido.

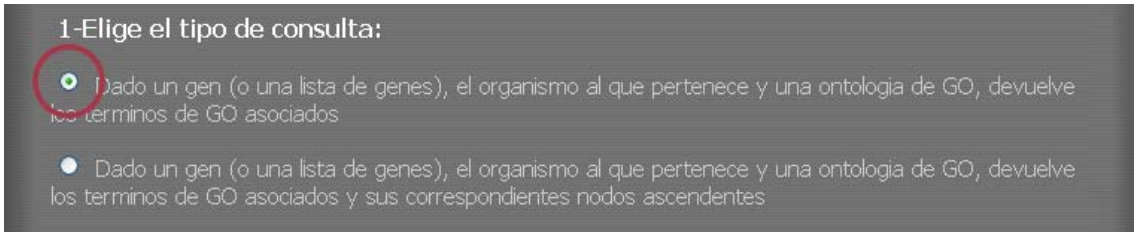
PRIMER TIPO DE CONSULTA.

El usuario quiere obtener todos los términos de GO únicos asociados a un determinado gen o grupo de genes de entrada. Para ello debe proporcionar los nombres de los genes, el organismo al que pertenece y una ontología de GO.

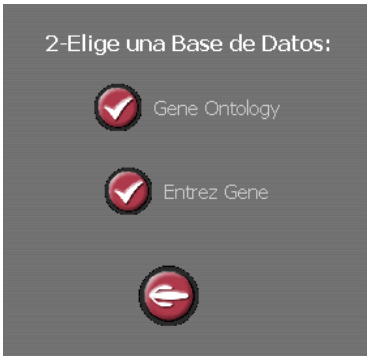
Por ejemplo: El usuario proporciona dos genes (AAT1, AAT2), del organismo Saccaromyces Cerevisiae y selecciona la ontología 'Biological Process'.

Vamos a explicar paso a paso como realizar la consulta:

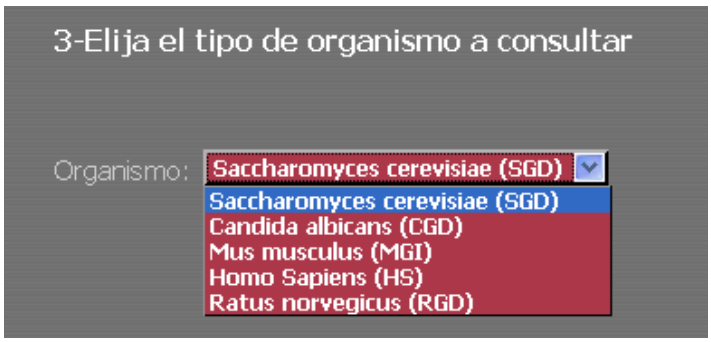
1) Seleccionar la consulta numero 1.



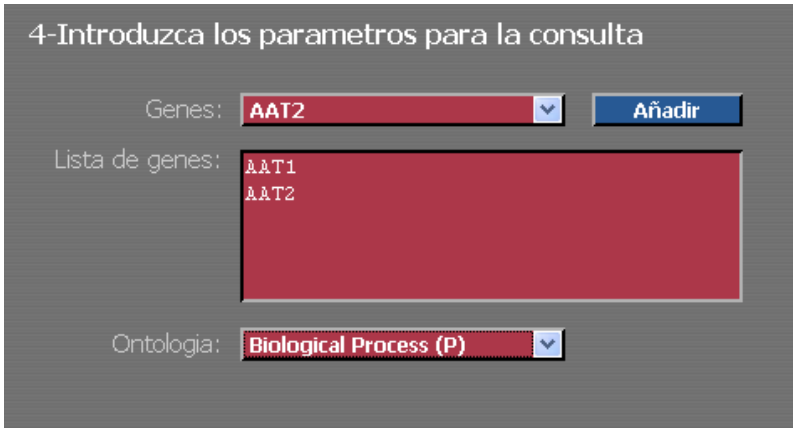
2) Seleccionar la base de datos a la que pertenecen los genes de entrada, pudiendo elegir Gene Ontology o Entrez Gene.



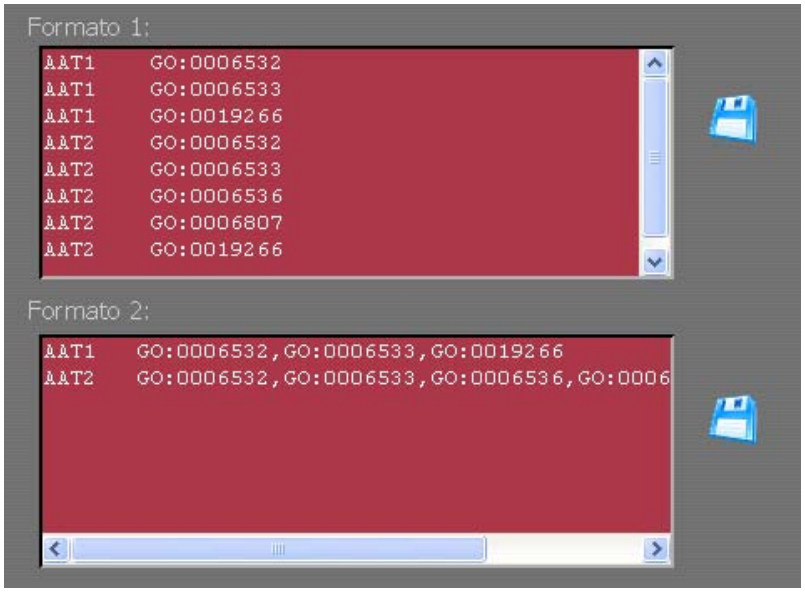
3) Selección del organismo. Organismos posibles.



4) Introducción de los genes a consultar y la ontología.



5) Obtención de resultados: Obtenemos los resultados en dos formatos, si deseamos guardarlos, seleccionamos el disquete y se abrirán en otra pantalla desde donde podremos guardarlos del modo habitual.

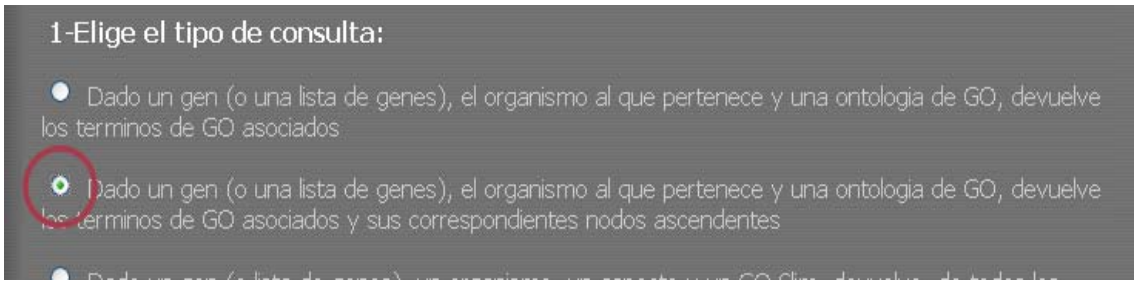


SEGUNDA CONSULTA.

El usuario quiere obtener todos los términos de GO únicos asociados a un determinado gen o grupo de genes de entrada y sus correspondientes nodos ascendentes. Para ello debe proporcionar los nombres de los genes, el organismo al que pertenece y una ontología de GO.

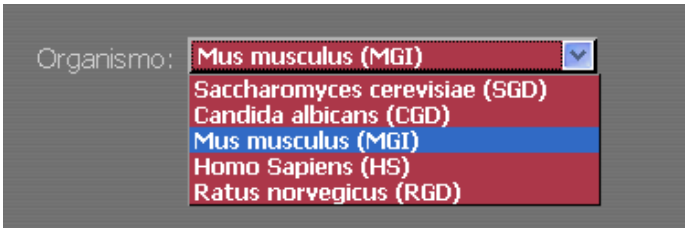
Por ejemplo: El usuario proporciona un gen (0610007C21Rik), del organismo Mus Musculus (MGI) y selecciona la ontología 'Molecular Function'.
Vamos a explicar paso a paso como realizar la consulta:

1) Seleccionar la consulta numero 2 CAPTURA6

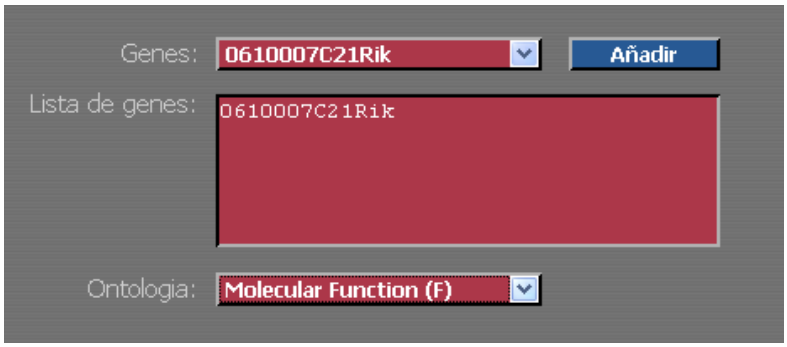


2) Seleccionar la base de datos a la que pertenecen los genes de entrada, pudiendo elegir Gene Ontology o Entrez Gene.

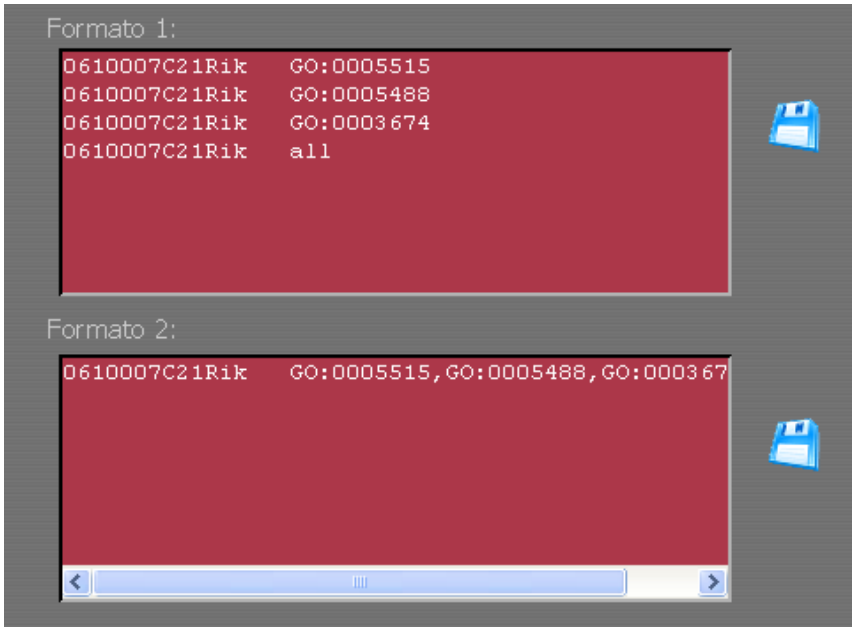
3) Selección del organismo.



4) Introducción de los genes a consultar y la ontología.



5) Obtención de resultados: Obtenemos los resultados en dos formatos, si deseamos guardarlos, seleccionamos el disquete y se abrirán en otra pantalla desde donde podremos guardarlos del modo habitual.



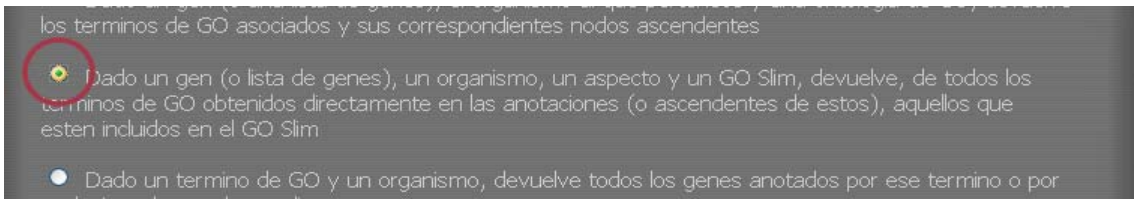
TERCER TIPO DE CONSULTA.

El usuario quiere obtener todos los términos GO de una lista de referencia (lista de GO Slims, predeterminada o introducida por el usuario) asociados a un determinado gen o grupo de genes de entrada y sus correspondientes nodos ascendentes. Para ello debe proporcionar los nombres de los genes, el organismo al que pertenece, una ontología de GO y una lista de GO Slims.

Por ejemplo: El usuario proporciona tres genes (ADE1, UBI3 y GSP1), del organismo Candida Albicans (CGD) y selecciona la ontología 'Biological Process' con la base de datos de Gene Ontology. Además, para este caso también tenemos los GO Slims.

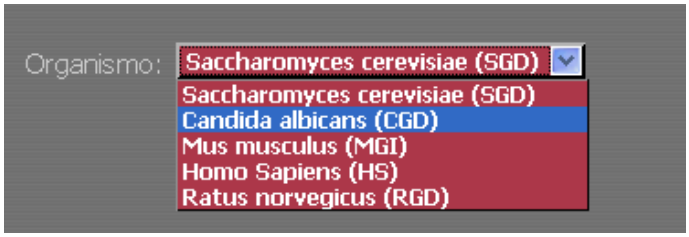
Vamos a explicar paso a paso como realizar la consulta:

1) Seleccionar la consulta numero 3.

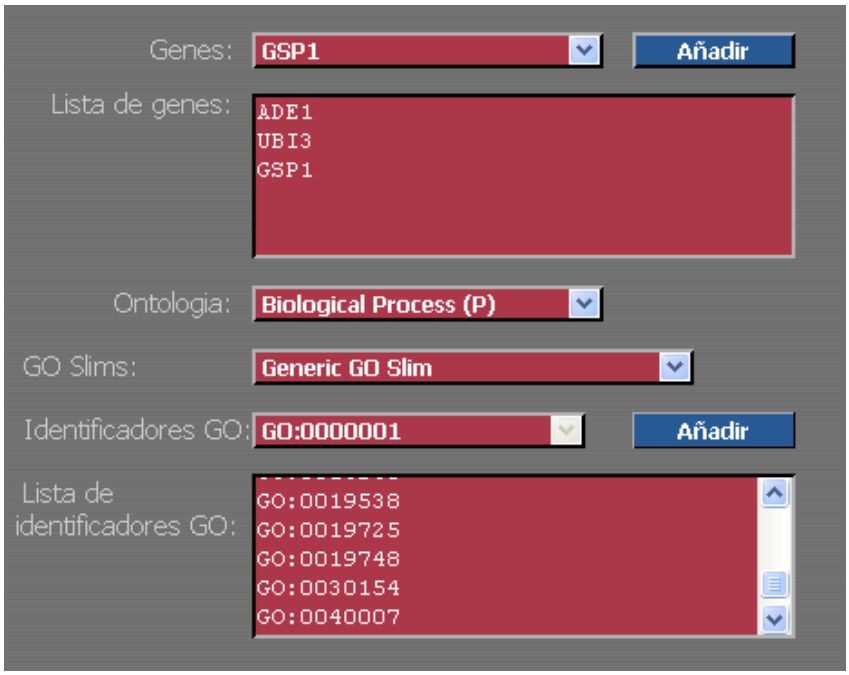


2) Como en los ejemplos anteriores, seleccionamos la base de datos a la que pertenecen los genes de entrada.

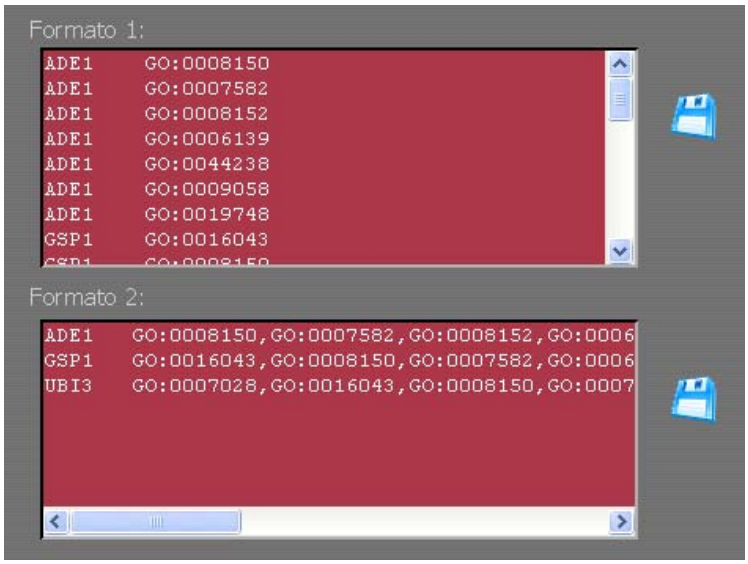
3) Selección del organismo.



4) Introducción de los genes a consultar, la ontología y la lista de GO Slims.



5) Obtención de resultados.



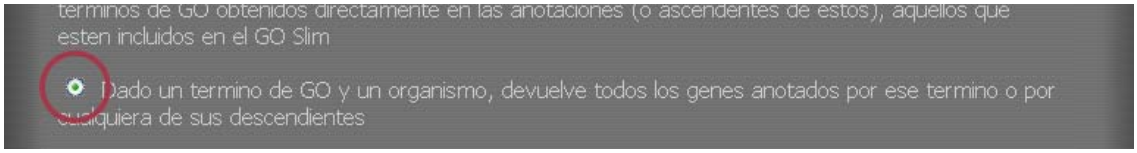
CUARTO CASO.

El usuario quiere, dado un término GO, obtener todos los genes anotados por ese término o por cualquiera de sus descendientes. Para ello debe proporcionar, además del término GO, el organismo.

Por ejemplo: El usuario proporciona como entrada el organismo Saccaromyces Cerevisiae, y el identidicador GO:0000011 en la base de datos de Gene Ontology.

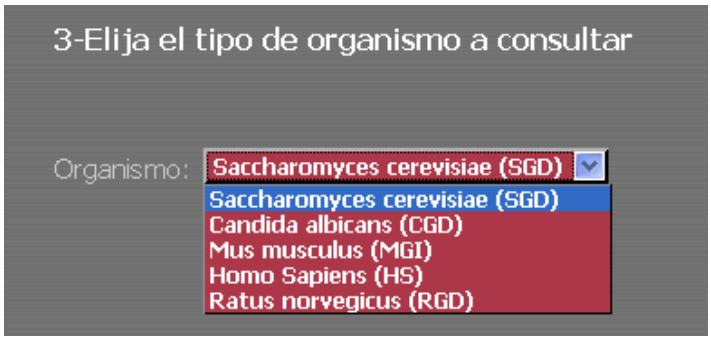
Vamos a explicar paso a paso como realizar la consulta:

1) Seleccionar la consulta numero 4.

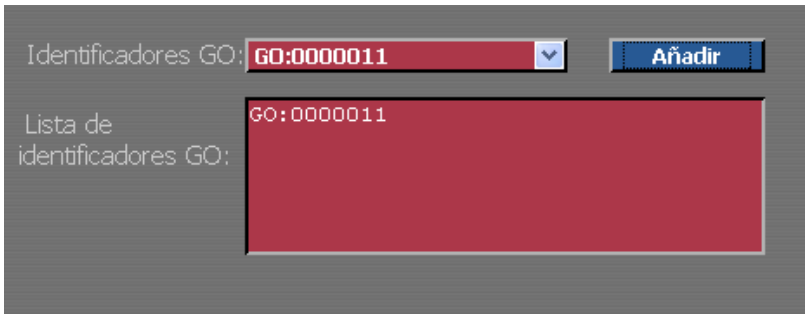


2) Seleccionar la base de datos a la que pertenecen los genes de entrada, pudiendo elegir Gene Ontology o Entrez Gene.

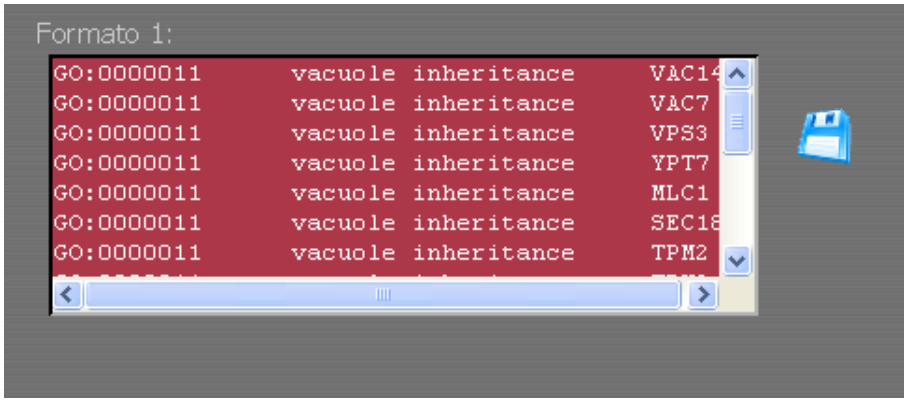
3) Selección del organismo. Organismos posibles



4) Introducción del término GO.



5) Obtención de resultados.



6. Conclusiones

Recordando los objetivos marcados al inicio del proyecto, descritos en el apartado 1.2, comentaremos a continuación cual es la situación una vez concluido este proyecto.

Como objetivo principal nos planteamos el desarrollo de un sistema basado en servicios web que permitiese acceder a las anotaciones funcionales de los organismos *Saccharomyces cerevisiae* y *Rattus norvegicus*. Además de estos, el sistema funciona para los organismos *Mus musculus*, *Candida albicans* y *Homo Sapiens.* Además se ha construido de tal forma que se pueden añadir nuevos organismos de forma sencilla.

En un principio el sistema se pensó para trabajar con la base de datos de Gene Ontology y posteriormente se amplió para su uso con la de Entrez Gene.

Otro de los objetivos propuestos y cumplidos era la creación de un mecanismo de actualización de la base de datos de Proyecto Genome: actualmente el mecanismo funciona actualizando todos los días las tablas de la base de datos así como los ficheros de anotaciones.

El último objetivo, la creación de un prototipo web, nos ha servido durante todo el desarrollo del proyecto como método de prueba de las distintas funcionalidades a medida que estas se iban terminando. Nos preocupamos de que, a pesar de ser un prototipo, resultase intuitivo, cómodo y útil a la hora de realizar consultas.

Como ampliaciones y trabajos futuros proponemos los siguientes:

- ampliación del sistema para soportar nuevos organismos y nuevas bases de datos de ámbito genómico.
- ampliación del número y los tipos de consultas posibles.
- adaptación de la aplicación para el funcionamiento en otro tipo de servidores.
- modificación del interfaz web existente para este proyecto.

7. Bibliografía

Hemos agrupado las fuentes por tecnologías y añadido las referencias web que consideramos interesantes y que nos han influido de uno u otro modo.

ONTOLOGIAS - GENE ONTOLOGY

- Enciclopedia libre Wikipedia
  http://es.wikipedia.org/

- Instituto de Investigaciones Biotecnológicas de Argentina
  Ontologías en Biología, por Fernán Agüero
  http://genoma.unsam.edu.ar/bioinformatica2005

- Página del proyecto Gene Ontology
  http://www.geneontology.org/

- OBO (Open Biomedical Ontologies)
  http://obo.sourceforge.net/

- Bioinformatics Packages Source Code Search
  http://b-src.cbrc.jp/markup/go/teaching_resources/tutorials

MySQL

- Bioinformatics Packages Source Code Search
  http://b-src.cbrc.jp/markup/go/teaching_resources/tutorials

- MySQL
  http://www.mysql.com/

- MySQL-Hispano
  http://www.mysql-hispano.org

- Gestor Gráfico SQLYog
  http://www.webyog.com/

SERVICIOS WEB - XML

- Servicios web XML
  Autores: Patrick Cauldwell, Rajesh Chawla, Vivek Chopra
  Editorial. Anaya Multimedia

- Understanding Web Services: XML, WSDL, SOAP, and UDDI
  Autor: Eric Newcomer
  Editorial: Independent Technology Guides

- Web Services Essentials (O'Reilly XML)
  Autor: Ethan Cerami
  Editorial: O'Reilly

- www.w3.org/2002/ws/

- www.conclase.net/

HTML

- La biblia, HTML 4.
  Autor: Molly E. Holzschlag
  Editorial: Anaya Multimedia, 2000

JAVASCRIPT

- Perl, CGI y JavaScript
  Editorial: Anaya Multimedia, 2000

- JavaScript. Programmer's Reference
  Autor: Cliff Wooton.
  Editorial: Wrox, 2001

- Guide to Building Intelligent Websites with JavaScript
  Autor: Nigel Ford
  Editorial: Wiley Computer, 1998

JSP

- Pro JSP
  Autor: Simon Brown & Cia
  Editorial: Apress, 3º Edición.

JAVA

- Java and XML
  Autor: Brett McLaughlin
  Editorial: O'Reilly, 1º edición, 2000

8. Glosario

Bioinformática: aplicación de las matemáticas y de las técnicas informáticas para comprender los problemas biológicos, normalmente creando o usando programas informáticos, modelos matemáticos o ambos

CSS: Hojas de Estilo en Cascada (Cascading Style Sheets), es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura.

Entrez Gene: base de datos que pertenece y es mantenida por el Centro Nacional de Información Biotectonológica (NCBI) y que contiene información genética de muchos organismos diferentes, incluidos virus, bacterias, plantas, humanos y otros animales.

Gene Ontology: ontología global que abraza tres ontologías generales (molecular function, biological proccess y cellular component), que pueden ser utilizadas independientemente y que describen atributos de elementos genéticos de cada uno de estos dominios de biología molecular.

Ontología: Estructuración jerárquica del conocimiento acerca de las cosas, mediante la clasificación en categorías de acuerdo con sus propiedades esenciales (o al menos relevantes y/o cognitivas).

SOAP: es el acrónimo de Simple Object Access Protocol. Protocolo ligero de mensajes XML que se usa para codificar la información de los mensajes de petición y respuesta de los Web Services que se envían a través de una red.

XML: (eXtensible Markup Language) es un lenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes.

Web Service: aplicación software identificada mediante una URI, cuyo interfaz (y uso) es capaz de ser definido, descrito y descubierto mediante artefactos XML, y soportar interacciones directas con otras aplicaciones software usando mensajes basados en XML y protocolos basados en Internet.