UNIVERSIDAD POLITECNICA DE VALENCIA

PROS

Centro de Investigación en Métodos de Producción de Software

Sistema de información genómico: Extensión del módulo de carga e integración de información genómica

Máster de Posgrado Oficial en Ingeniería del Software, MétodosFormales y Sistemas de Información

Valencia, 1 de Julio de 2013

Jorge Guerola Martínez
jorguema@ei.upv.es

Director: Óscar Pastor López
opastor@pros.upv.es

Co-director: Mercedes R. Fernández Alcalá
mfernandez@pros.upv.es

ÍNDICE GENERAL

Contenido

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1 Motivación

Durante los últimos años los sistemas de información constituyen uno de los principales ámbitos de estudio en el área de organización de empresas ya que el entorno donde las compañías desarrollan sus actividades es cada vez más complejo. La creciente globalización, el proceso de internacionalización y la rapidez en el desarrollo de las tecnologías de información originan que la información se convierta en un uno de los principales recursos de las organizaciones. Se ha comenzado a comprender que la información no es sólo un subproducto, sino que alimenta a los negocios y puede ser uno de los tantos factores críticos para la determinación del éxito o fracaso.

Los Sistemas de Información (SI) han cambiado la forma en que operan las organizaciones actuales. A través de su uso se logran importantes mejoras, pues automatizan los procesos operativos y suministran una plataforma de información útil y de calidad necesaria para la toma de decisiones.

Una implicación ampliamente aceptada es que el desarrollo de sistemas de información (SI) de calidad exige el uso de metodologías basadas en el modelado conceptual. El uso de modelos conceptuales facilita una visión general en un alto nivel de abstracción, permitiendo conocer y comprender el dominio del problema antes de abordar su solución.

Los modelos conceptuales son utilizados en muchos contextos y ámbitos de aplicación. Sin embargo, a pesar de los beneficios que aportan, muchos dominios permanecen alejados de su uso, ya sea por su complejidad, los grandes volúmenes de datos o la difícil integración, entre otros. Un ejemplo de este tipo de dominio es el de la bioinformática. El esfuerzo de desarrollo realizado en este ámbito es para problemas concretos, prestando poca atención al desarrollo de sistemas de información que proporcionen información genómica de calidad.

La Genómica, disciplina que estudia el genoma de los organismos, es un campo en constante evolución. Desde 1970, las técnicas de secuenciación, alineamiento de secuencias y análisis biológico han avanzado rápidamente produciéndose una gran cantidad de datos en los laboratorios. Este ingente volumen de información disponible ayuda, obviamente, al investigador en su trabajo, pero al mismo tiempo, es un hecho que su constante crecimiento dificulta cada vez más la búsqueda de la información más adecuada en cada momento. Esta dificultad se incrementa debido a que la información está dispersa en numerosos repositorios, sitios web, bancos de datos, ficheros públicos, etc, por lo que la tarea de encontrar alguna información dentro de este caos se convierte en una tarea tediosa para los biólogos, que incluso a veces puede llegar a ser uno objetivo imposible de alcanzar. Si a esta situación, de crecimiento constante del volumen de datos y del número de repositorios disponibles, se añade que la misma disciplina, es decir los conceptos biológicos sobre los que se asienta, están

en continua evolución, la necesidad de abordar la construcción de sistemas de información genómica desde una aproximación metodológica se convierte en una necesidad.

Dada la situación actual, es necesaria la integración de la información genómica, existente en diferentes repositorios de información dispersos, en un sistema de información, flexible a la evolución del dominio y al incremente de repositorios. De esta forma se posibilita la búsqueda, identificación, carga y explotación de datos genómicos. La búsqueda, identificación, carga y explotación del Adenomatous polyposis coli (APC), gen del genoma humano, es el objetivo de este trabajo.

1.2 Planteamiento del problema

El coste temporal empleado en la búsqueda de la información genomica es muy elevado. Esta búsqueda implica explorar los diferentes repositorios genómicos, identificar la información relevante y validar que la información sea correcta.

La relevancia de la información genómica asociada al gen Adenomatous polyposis coli (APC), desde el punto de vista biológico, implica que unas de las búsquedas principales realizadas en los repositorios genómicos esten relacionada con este gen.

La información genómica junto al conjunto de variaciones asociadas de este gen se encuentran distribuidas en diferentes repositorios. Al ser de gran importancia es necesario proveer a investigadores y genetistas de esta información genómica, centralizando toda la información relevante en un único sistema de información genómico.

Ofrecer información de calidad es esencial en el ámbito de la genómica, pora ello, toda la información debe de estar referénciada del repositorio en donde ha sido extraída, además se debe abordar la posible existencia de múltiples referéncias bibliográficas, incorporando un valor adicional a la información.

1.3 Objetivos

El objetivo principal de esta tesis es integrar toda la información asociada a los cromosomas del genoma humano al mismo tiempo que se integra en este nuevo modelo conceptual, la carga del gen Adenomatous polyposis coli (APC) de los diferentes repositorios de información, solucionando la falta de información cromosómica e integrando la información relevante del gen APC.

Para alcanzar el objetivo principal de la tesis se cumplirán los siguientes objetivos específicos:

1. Estudiar el modelado conceptual.

2. Identificar la información cromosómica relevante.

3. Incorporar la información cromosomica para la evolución del modelo conceptual.

4. Integración del gen APC en el modelado conceptual mediante la metodología SILE.

1.4 Estructura de la tesis

A fin de cumplir con los objetivos descritos, la tesina se estructura como se detalla a continuación.

En el primer capítulo introducimos cual es la problemática actual con la información genómica, el planteamiento del problema que se encargará de resolver esta tesis y la metodología a emplear para alcanzar los objetivos propuestos.

En el capítulo 2, se presenta el estado del arte, en donde se comenta el estado de la cuestión y los principales trabajos existentes relacionados con el modelado conceptual en el ámbito genómico, además de la importancia de este contexto. Por otra parte aboraremos una serie de de requisitos y conceptos biológicos con el objetivo de comprender mejor los aspectos más relevantes de esta ciencia y de los que se habla en este trabajo.

En el tercer capítulo abordaremos el diseño y desarrollo del modelo conceptual del genoma humano, realizando una explicación de las diferentes vistas, incluyendo la vista de información cromosómica, y la generación de la base de datos partiendo de este modelo.Tras la explicación del modelado conceptual, en el capítulo 4, comentaremos que es la metodología SILE y para que se utiliza.

En el capítulo 5 veremos cómo aplicar la metodología SILE sobre el gen Adenomatous polyposis coli. Para ello veremos tanto su información general como detallada, los repositorios en los que se ha encontrado información del gen y de las variaciones y los repositorios relevantes para el estudio. Analizaremos el proceso de carga genómico, para ello veremos que para cargan un gen en el sistema de información generado es necesario realizar previamente una carga estructural del gen, y posteriormente, la extracción, transformación y carga de las variaciones. Finalizando este capítulo resaltaremos la aportación realizada en esta tesis basada

es una mejora realizada en el proceso de carga de los genes para soportar múltiples referencias bibliográficas.

Por último y para finalizar este trabajo, en el capítulo 6, desarrollaremos las conclusiones extraídas del esta tesis, así como la posibilidad de trabajos futuros para la optimización y mejora del proceso de carga.

2. ESTADO DEL ARTE

El uso del modelado conceptual en el campo de la bioinformática ha ido en constante evolución. Es cierto que existen numerosos repositorios de datos que almacenan información genómica, pero la mayoría de ellos requieren mejoras de almacenamiento y no es usual encontrar un riguroso y sólido modelo conceptual como base de ellos.

Existen algunas propuestas realizadas en este ámbito. El pionero fue Paton [2-4] que introdujo los primeros trabajos sobre el modelado conceptual del genoma desde diferentes perspectivas. Paton presentó modelos que describían el genoma de la célula eucariota, la interacción entre proteínas, el transcriptoma, etc., sin embargo su trabajo no tuvo una clara continuación.

Por otro lado Ram et al [5] también aplicaron principios de modelado conceptual pero en el contexto de las proteínas. La consulta de datos voluminosos y de estructura compleja, como es el caso de las proteínas 3D, requiere el uso de modelos expresivos que soporten explícitamente y capturen la semántica de este tipo de datos. En su trabajo Ram muestra cómo la comparación y búsqueda en la estructura de una proteína en 3D se facilita con el modelado conceptual. A pesar de ser un dominio “pequeño”, se demuestra que el modelado conceptual ayuda a manejar datos de manera efectiva.

Continuando con otro tipo de propuestas existentes basadas en técnicas de modelado conceptual, podemos destacar también un conjunto importante de implementaciones bioinformáticas favorablemente aceptadas por la comunidad científica en las que en mayor o menor grado también han sido utilizadas estas técnicas. Un claro ejemplo de esto es el trabajo realizado por Garwood [3] en el cual se introduce una aproximación dirigida por modelos para la generación parcial de interfaces de usuario cuyo objetivo es realizar búsquedas en repositorios de datos bioinformáticos. Este trabajo demuestra que los modelos conceptuales pueden ser usados para generar aplicaciones futuras y no únicamente para representar un dominio. Cuando comparamos este trabajo con nuestro modelado conceptual [1], cabe destacar que las técnicas de modelado que utilizaban era únicamente para solventar una parte del gran problema al que se enfrentaban, las interfaces de usuario, mientras que nosotros proporcionamos una amplia y unificada vista de modelado conceptual.

Otro enfoque propuesto para representar conceptos relacionados con el genoma, son los intentos de unificación de términos realizados por expertos en el campo de las ontologías. Un ejemplo de este tipo de representación la proporciona GeneOntology [6], iniciativa que nació con el objetivo de estandarizar la representación de los genes y sus atributos. El proyecto proporciona un vocabulario controlado de términos para describir todas las características de los genes y de los datos anotados en el genoma a través de una herramienta de acceso a ellos. A pesar del esfuerzo realizado, este tipo de solución que proporciona va más orientada a solucionar un problema en concreto que a solucionar una situación global que afecta al mundo de la bioinformática.

Partiendo de la necesidad de un sistema de información en el campo de la bioinformática y basándose en las propuestas anteriores, surge el modelado conceptual del genoma humano [1]. Este modelo conceptual fue realizado por el departamento de PROS en la Universidad Politécnica de Valencia y representa toda la información relevante en el ámbito de la bioinformática. En esta aproximación se explican las diferentes vistas que representan el genoma humano y la relación entre ellas.

En el ámbito de la bioinformática el conocimiento adquirido desde el inicio de los tiempos es mínimo en comparación con todo lo que está por descubrir. Esto implica un campo en constante evolución, y por lo tanto, es necesario adaptarse a él. En este proceso de evolución muchas técnicas de análisis del genoma humano van quedándose obsoletas y son reemplazadas por nuevas técnicas que proporcionan resultados más certeros. Para obtener resultados más concretos se necesita información genómica más detallada. Esta nueva información requerida está relacionada con la parte de los cromosomas del genoma humano. Por lo tanto, al estar en un ámbito en constante evolución, nuestro modelo conceptual también debe estarlo, con el fin de adaptarse a las nuevas exigencias. Esta evolución requiere la adición de la vista cromosómica del genoma humano con el fin de mantener un modelo actualizado a acorde a las necesidades biológicas actuales.

2.1 Requisitos biológicos

La Genética es una ciencia que nace como una rama de la biología con el objetivo de comprender la herencia biológica que se transmite de generación en generación.. La Genética es la rama de la Biología que trata de la herencia y de su variación. La herencia se refiere a que la descendencia tiende a asemejarse a sus padres, basándonos en el hecho de que nuestro aspecto y función biológica, es decir, nuestro fenotipo, viene determinado en gran medida por nuestra constitución genética, es decir, nuestro genotipo.

La Genética estudia la forma en que las características de los organismos vivos, sean éstas morfológicas, fisiológicas, bioquímicas o conductuales, se transmiten, se generan y se expresan, de una generación a otra, bajo diferentes condiciones ambientales. Éstas características pueden ser tanto estéticas, fisiológicas e incluso de comportamiento. Así pues se trata de estudiar cómo se transmiten éstas características a futuras generación y porque varían.

En 1865, Gregor Mendel, al que podríamos denominar el padre de la genética ya que realizó los primeros trabajos existentes esta ciencia, describió por medio de trabajos llevados a

cabo con diferentes variedades del guisante, las hoy llamadas leyes de Mendel que rigen la herencia genética y que más tarde fueron ampliadas y generalizadas a un gran número de organismos vivos:

I. Ley de Mendel o principio de la uniformidad: Establece que si se cruzan dos razas puras para un determinado carácter, los descendientes de la primera generación serán todos iguales entre sí fenotípica y genotípicamente, e iguales fenotípicamente a uno de los progenitores (de genotipo dominante), independientemente de la dirección del cruzamiento
II. Ley de Mendel o principio de la segregación: Esta ley establece que durante la formación de los gametos, cada alelo de un par se separa del otro miembro para determinar la constitución genética del gameto filial. Esto significa que en las células somáticas, un alelo proviene de la madre y otro del padre
III. Ley de Mendel o principio de la combinación independiente: que diferentes rasgos son heredados independientemente unos de otros, no existe relación entre ellos, por lo tanto el patrón de herencia de un rasgo no afectará al patrón de herencia de otro. Sólo se cumple en aquellos genes que no están ligados (es decir, que están en diferentes cromosomas) o que están en regiones muy separadas del mismo cromosoma.

El principal objeto de estudio de la genética es el ADN o ácido desoxirribonucleico, descrubierto por James Watson, Francis Crick y Maurice Wilkins en 1951. El ADN es un ácido nucleico que contiene instrucciones genéticas usadas en el desarrollo y funcionamiento de todos los organismos vivos conocidos y algunos virus, y es responsable de su transmisión hereditaria. El papel principal de la molécula de ADN es el almacenamiento a largo plazo de información. El cuerpo humano está formado por 10 billones de células que se conocen como las unidades funcionales de los seres vivos. Cada una de ellas, posee una zona llamada núcleo donde se almacena la información genética en forma de ADN. Este ácido es la molécula que controla todos los procesos celulares como la alimentación y la reproducción celulares o la transmisión de caracteres de padres a hijos.

Para abordar toda la información relevante del genoma realizaremos una clasificación acorde a su funcionalidad: Se comenzará con una introducción a la estructura genómica, resaltando sus elementos principales, seguido del ciclo de vida del ADN y concluyendo con la definición de una variación y las consecuencias que aporta.

2.1.1 Estructura genómica

En este apartado mencionaremos de forma general la estructura genómica, el proceso de replicación, transcripción y traducción y por último la información relevante de las variaciones genéticas.

2.1.1.1 El ADN

La molécula de ADN se encuentra formada por dos cadenas muy largas enrolladas entre sí formando una estructura helicoidal alrededor de un eje que da lugar a una doble hélice parecida a una escalera de caracol. La parte lateral de esta escalera está formada por fosfatos y azucares orientados hacia el exterior de la molécula y los peldaños son pares de bases. En esta estructura, la adenina se empareja con la timina (A-T, T-A) y la citosina se empareja con la guanina (C-T, T-C). Ya que el esqueleto azúcar-fosfato es siempre igual, la manera de escribir la información genética se realiza mediante un alfabeto de 4 letras en el cual se tiene en cuenta el tipo de nucleótidos y el orden en que se disponen. Esta disposición de la información de manera habitual es denominada secuencia. Esta molécula de ADN tiene la capacidad de desdoblarse y dar lugar a otra molécula idéntica, así es como pasa la información a sus hijos. El ADN es la base de la herencia.

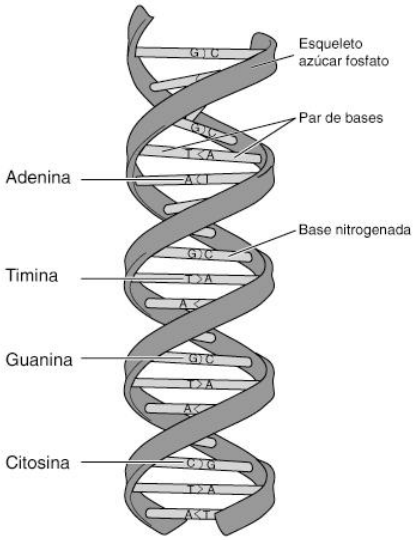


Ilustración 1-ADN

Si estirásemos el ADN, llegaría a medir hasta 1,8 metros, es decir, unas 300000 veces más que el núcleo. Para evitar este problema, el ADN esta plegado formando unas estructuras denominadas cromosomas. Cada cromosoma es una única molécula de ADN, que a su vez está formada por miles de nucleótidos.

2.1.1.2 Cromosoma

Los cromosomas son pequeños cuerpos en forma de bastoncillos en que se organiza la cromatina del núcleo celular durante las divisiones celulares. Son segmentos largos de ADN que se encuentran en el núcleo de las células.

Los cromosomas vienen en pares. Normalmente, cada célula en el cuerpo humano tiene 23 pares de cromosomas (46 cromosomas en total), de los cuales la mitad proviene de la madre y la otra mitad del padre. De todos estos un par son cromosomas sexuales (determinan el sexo del sujeto) y 44 son autosómicos (no sexuales).

En la descendencia genética los cromosomas sexuales X e Y son los encargados de determinar el sexo. Las mujeres tienen 2 cromosomas X y los hombres tienen un cromosoma X y uno Y. La madre siempre le aporta un cromosoma X al hijo, mientras que el padre puede contribuir ya sea con un cromosoma X o con un cromosoma Y determinando el sexo.

En un cromosoma se encuentran muchos elementos, entre ellos los genes. Por ejemplo, en cada célula del cuerpo humano hay aproximadamente 30.000 genes y cada uno de ellos ocupa en el cromosoma una posición determinada llamada locus [7].

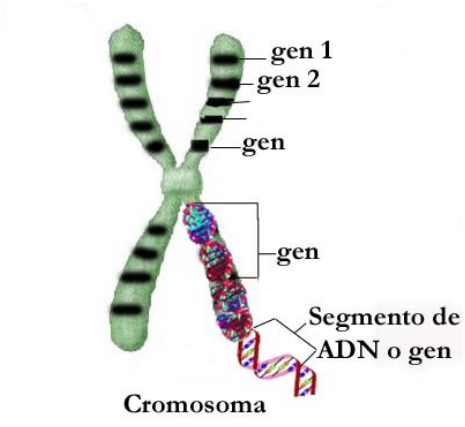


Ilustración 2-Cromosoma

2.1.1.3 Gen

El gen es considerado la unidad de almacenamiento de información genética y unidad de la herencia, pues transmite esa información a la descendencia. Los genes se disponen a lo largo de ambas cromátidas de los cromosomas y ocupan, en el cromosoma, una posición determinada llamada locus. El conjunto de genes de una especie, y por tanto de los cromosomas que los componen, se denomina genoma. Los genes están localizados en los cromosomas en el núcleo celular.

Un gen es una secuencia lineal de nucleótidos de ADN que es esencial para una función específica, da lugar a un ARN (ácido ribonucleico) a través de un proceso de transcripción y lleva la información para sintetizar una proteína. Como el ADN, el ARN también está formado por una cadena de nucleótidos, pero a diferencia de éste, la molécula de ARN contiene un átomo de oxigeno que el ADN no tiene y contiene la base de uracilo U en lugar de la timina T. La secuencia de bases presente en el ARN determina la secuencia de aminoácidos de la proteína por medio del código genético. Es importante resaltar que, si bien el ADN es donde se almacena la información genética de un organismo, las proteínas son las que ejecutan dicha información porque son las moléculas esenciales para todos los aspectos de estructura y actividad celular.

No todos los genes codifican proteínas sino que algunos de ellos cumplen su función en forma de ARN, como por ejemplo regular post-transcripcionalmente otros genes. Entre estos encontramos genes de ARN transferente, micro ARN, ARN ribosómico, ribozimas y otros ARN pequeños de funciones diversas.

2.1.2 Replicación, transcripción y traducción del ADN

El proceso de replicación de ADN o herencia celular es llevado a cabo mediante la Replicación, transcripción y traducción. Estos tres procesos son necesários abordar la herencia genética y la replicación de la información genética en una célula.

3.2.1 Replicación

El proceso de replicación de ADN es el mecanismo que permite al ADN duplicarse, es decir, sintetizar una copia idéntica [8]. De esta manera de una molécula de ADN única, se obtienen dos o más "clones" de la primera. Esta duplicación del material genético se produce de acuerdo con un mecanismo semiconservativo, lo que indica que las dos cadenas

complementarias del ADN original, al separarse, sirven de molde cada una para la síntesis de una nueva cadena complementaria de la cadena molde, de forma que cada nueva doble hélice contiene una de las cadenas del ADN original. Gracias a la complementación entre las bases que forman la secuencia de cada una de las cadenas, el ADN tiene la importante propiedad de reproducirse idénticamente, lo que permite que la información genética se transmita de una célula madre a las células hijas y es la base de la herencia del material genético.

3.2.2 Transcripción

La transcripción del ADN es el primer proceso de la expresión génica, mediante el cual se transfiere la información contenida en la secuencia del ADN hacia la secuencia de proteína utilizando diversos ARN como intermediarios.

Cuando comienza, en la fase de iniciación, el ADN se separa para poder ser copiado ya que ha de ser asequible para la enzima ARN-polimerasa. En la siguiente fase, fase de elongación, las materias primas que forman la molécula de ARN: ATP, GTP, CTP, UTP, quedan enlazadas a lo largo de una cadena sencilla de ADN. Durante la fase de maduración las secuencias intrónicas, aquellas que no contienen información para la síntesis de proteínas, son eliminadas de la secuencia, dando lugar a una secuencia formada únicamente por exones que forman la región codificante del gen y transportan la información para producir la proteína.

Este proceso de retirada de los intrones y conexión de los exones se llama Splicing y da lugar al ARNm (ARN mensajero) maduro. Es importante mencionar que un mismo gen puede producir diferentes proteínas gracias al fenómeno conocido como Splicing Alternativo en el que algunos exones o parte de ellos pueden ser eliminados junto con los intrones que los flanquean y algunos intrones o parte de ellos pueden no ser eliminados durante el proceso. De esta manera se crean diversos ARNm que son traducidos a su vez en distintas proteínas. Cabe destacar que este Splicing Alternativo, no es de ninguna manera un proceso aleatorio sino que ha evolucionado de manera que las diferentes proteínas así creadas sean todas funcionales.

Este proceso de traducción se produce en el núcleo de la célula y dará lugar a la molécula de ARNm. Tras la formación de dicha molécula, ésta se desplazará hasta el citoplasma de la célula a través de los poros de la membrana nuclear para participar en el proceso de traducción o síntesis de proteínas.

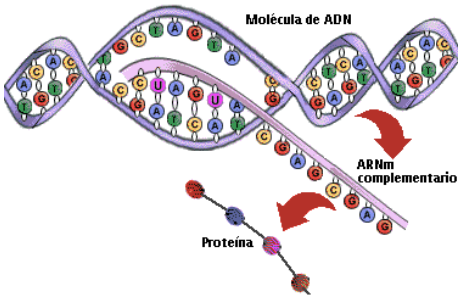


Ilustración 3-Transcripción

3.2.3 Traducción

La traducción es el paso de la información transportada por el ARN-m a proteína. Con la información genética contenida en el ARNm el siguiente paso deberá ser la traducción en el citoplasma mediante el ribosoma. Un término utilizado en la traducción es el codón, grupo de tres nucleótidos adyacentes que codifican un aminoácido.

La activación de los aminoácidos para formar los complejos de transferencia es el paso previo necesario para que pueda comenzar la traducción, y consiste en la unión de cada aminoácido a su ARN-t específico mediante la intervención de un enzima, la aminoacil-ARN-t sintetasa y el aporte de energía del ATP.

Una vez activados los aminoácidos y formados los complejos de transferencia (ARN-t cargados con el aminoácido correspondiente) ya puede comenzar la síntesis de la cadena polipeptídica y la incorporación de los aminoácidos. En este proceso se pueden distinguir tres fases diferentes:

I. Iniciación de la cadena polipeptídica: El ARNm se une a la subunidad menor de los ribosomas. A éstos se asocia el aminoacil-ARNt, gracias a que el ARNt tiene en una de sus asas un triplete de nucleótidos denominado anticodón, que se asocia al primer codón del ARNm según la complementariedad de las bases. A este grupo de moléculas se une la subunidad ribosómica mayor, formándose el complejo ribosomal o complejo activo.

II. Elongación de la cadena polipeptídica: La elongación o crecimiento de la cadena polipeptídica tiene lugar en esencia mediante la formación de enlaces péptídicos entre los aminoácidos sucesivos. Se repite tantas veces como aminoácidos posea el polipétido sintetizado menos uno (excepto el primero, metionina).

III. Terminación de la cadena polipeptídica: Los codones UAA, UAG y UGA son señales de paro que no especifican ningún aminoácido y se conocen como codones de terminación; determinan el final de la síntesis proteica

Cabe destacar que cada nucleótido tiene cuatro posibles valores (UCAG) lo cual supone que en la síntesis de la cadena polipeptídica, al unirse con el triplete de nucleótidos, pueden llegar a existir 64 posibles aminoácidos.

	U		C		A		G		
U	UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys	U
	UUC	Phe	UCC	Ser	UAC	Tyr	UGC	Cys	C
	UUA	Leu	UCA	Ser	UAA	*Stop*	UGA	*Stop*	A
	UUG	Leu	UCG	Ser	UAG	*Stop*	UGG	Trp	G
C	CUU	Leu	CCU	Pro	CAU	His	CGU	Arg	U
	CUC	Leu	CCC	Pro	CAC	His	CGC	Arg	C
	CUA	Leu	CCA	Pro	CAA	Gln	CGA	Arg	A
	CUG	Leu	CCG	Pro	CAG	Gln	CGG	Arg	G
A	AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser	U
	AUC	Ile	ACC	Thr	AAC	Asn	AGC	Ser	C
	AUA	Ile	ACA	Thr	AAA	Lys	AGA	Arg	A
	AUG	Met	ACG	Thr	AAG	Lys	AGG	Arg	G
G	GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly	U
	GUC	Val	GCC	Ala	GAC	Asp	GGC	Gly	C
	GUA	Val	GCA	Ala	GAA	Glu	GGA	Gly	A
	GUG	Val	GCG	Ala	GAG	Glu	GGG	Gly	G

Ilustración 4-Código de proteínas

2.1.3 Variación genética

Las células poseen una maquinaria muy sofisticada y precisa que permiten realizar copias perfectas de una molécula de ADN, existen incluso diversos sistemas que desechan aquellas copias que hayan sido finalizadas correctamente. No obstante, en ocasiones ocurren ciertos fallos que son desapercibidos por dichos mecanismos de reparación y estos cambios no son eliminados pudiendo llegar a cambiar la información que se transmite a la molécula de ARN.

Un simple cambio en la secuencia de un gen, puede generar desde la indiferencia hasta las consecuencias más drásticas. Un cambio o variación en la secuencia de ADN es el responsable de las diferencias fenotípicas (rasgos físicos o conductuales). Esto supone las diferencias encontradas en cada uno de nosotros, por ejemplo diferente color de ojos, altura, color etc. En ocasiones un cambio o variación puede que no se manifieste mediante fenotipos sino mediante genotipos, como podría ser el cambio o variación es la causante de ciertas enfermedades como el cáncer o la fibrosis quística que puede llegar a ocasionar incluso la muerte. Esto es debido a que una variación en la reproducción de una célula puede interrumpir la actividad normal de un gen dejando a éste inerte de realizar sus funciones originales.

Las mutaciones genéticas localizadas en el ADN suceden en una posición concreta y se pueden clasificar dependiendo el tipo de cambio:

- Sustituciones. En ocasiones también son llamadas mutaciones puntuales. Uno o más nucleótidos en la secuencia de ADN son sustituidos por otra secuencia de nucleótidos de menor, igual o mayor tamaño.
- Inserciones. Son aquellas en las que una o varias bases de nucleótidos adicionales se introducen en la secuencia de ADN.
- Deleciones o borrados. Ocurren cuando una base o varias bases de nucleótidos de la secuencia se eliminan.
- Inversiones. Cuando una sección del cromosoma se encuentra en la secuencia de forma invertida.
- Translocaciones. Se originan cuando un segmento de cromosoma se intercambia o se traspasa a otro cromosoma.

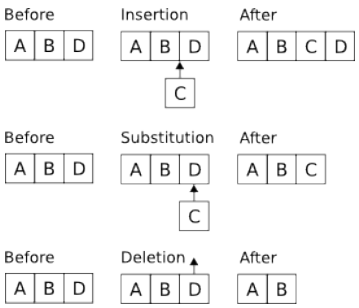


Ilustración 5-Variaciones

Cuando se indica que se ha realizado una variación debe indicarse obligatoriamente la posición. La posición donde se ha producido variación puede ser la posición respecto al o respecto al cromosoma. La posición en la que se haya producido la variación es un detalle muy importante, ya que si una inserción, un borrado o una sustitución de diferente tamaño ocurre dentro de un gen o cromosoma, se produce en una posición concreta de la secuencia de nucleótidos.

3. MODELADO CONCEPTUAL

Dada la necesidad de un sistema de información genómico, en el año 2008 se inicia una línea de investigación sobre el desarrollo y evolución del genoma que pretende la realización un sistema de información en el ámbito de la bioinformática. Esta propuesta, llevada a cabo en el Centro de Investigación ProS de la UPV, pretende solventar todos los problemas en este ámbito y proporcionar una metodología de trabajo efectiva a biólogos. Desde el inicio, varias son las versiones del modelo conceptual del genoma humano que se han desarrollado dentro del grupo de investigación. La primera de ellas [9], comúnmente conocida como la versión 1, modelaba con precisión las diferentes vistas para la representación del genoma humano. Con el paso del tiempo nueva información era necesaria y es por esto que el modelo inicial evolucionó a versiones posteriores [10, 11] comúnmente conocidas como versión 2. Esta segunda versión del modelo incorporaba aspectos relacionados con las mutaciones y la relación genotipo-fenotipo asociadas, además de incluir nuevos vistas y pathways.

Un modelo es una representación simplificada de la realidad elaborado para facilitar su compresión y estudio permitiendo identificar las distintas variables en el contexto y las relaciones entre ellas. Un modelo debe representar la realidad con la mayor fidelidad posible y mantener un balance entre precisión y complejidad. Así un modelo muy simplificado se aleja de la realidad, pero se acerca a la generalidad y es de fácil manejo; por el contrario, un modelo muy preciso se encuentra muy próximo a la realidad concreta, pero su utilización puede resultar compleja. El predominio de una u otra de estas características dependerá de la utilización que queramos hacer del modelo.

El objetivo de un modelo es abstraer el problema mediante un modelo independiente de plataforma, es decir, ajeno a cualquier rasgo de implementación o tecnología. Al ser un modelo independiente de plataforma se modela una vez y permite ser generado en diversas plataformas.

La evolución de la tecnología informática y su uso extendido han convertido a los sistemas de información en el pilar básico de las organizaciones. Un modelo conceptual y un sistema de información son términos condicionalmente ligados. Para obtener un sistema de información en un ámbito específico es necesario representar toda la información de forma estructural y organizada. Esta representación de la información implica el diseño un modelo conceptual.

Para el desarrollo del sistema de información para el ámbito genómico es muy importante tenerse en cuenta la complejidad del dominio, la gran evolución y la heterogeneidad que presenta. La constante evolución en este campo es un factor que se ha de tener presente, ya que nuestro modelo propuesto, debe estar actualizado en todo momento con el fin de adaptarse a las nuevas exigencias.

La evolución en este entorno demanda que el proceso de secuenciación utilizado en este ámbito esté en cosntante evolucion es decir, el proceso de obtención del ADN del paciente para realizar un diagnóstico genético. Inicialmente los métodos de secuenciación realizados se basaban en obtener diversas muestras de los genes del paciente. Las nuevas máquinas de secuenciación y el conocimiento sobre este campo permiten obtener mayor información en la secuenciación del ADN y esto implica la evolución de análisis genético a análisis cromosómico, el cual incluye muestras a nivel cromosómico genético.

Estos nuevos avances suponen nuevas exigencias para el modelo conceptual ya que se ha de integrar la nueva información necesaria en el modelo. Estos nuevos requisitos de información están relacionados con la versión del genoma más reciente, las secuencias cromosómicas y la partición de éstos en elementos del cromosoma, con el fin de reducir la gran longitud de secuencia, la secuencia del gen y el posicionamiento tanto a nivel génico como cromosómico.

Conociendo la nueva información a integrar en nuestro modelo conceptual y supervisada por un grupo de expertos en el centro de investigación ProS, se ha realizado una evolución del modelo conceptual genómico anterior a el nuevo modelo conceptual genómico versión 3. Para una mejor comprensión de este modelo se divide en cinco vistas, todas ellas relacionadas entre sí, pero lo suficientemente independientes de las demás como para poder definirlas por separado sin perder información. Dichas vistas son: la vista estructural, la vista de transcripción, la vista de variaciones, la vista de rutas metabólicas y la vista de bibliografía y fuentes de datos. Para destacar las clases UML de intersección entre dichas vistas, éstas aparecen sombreadas en los diagramas.

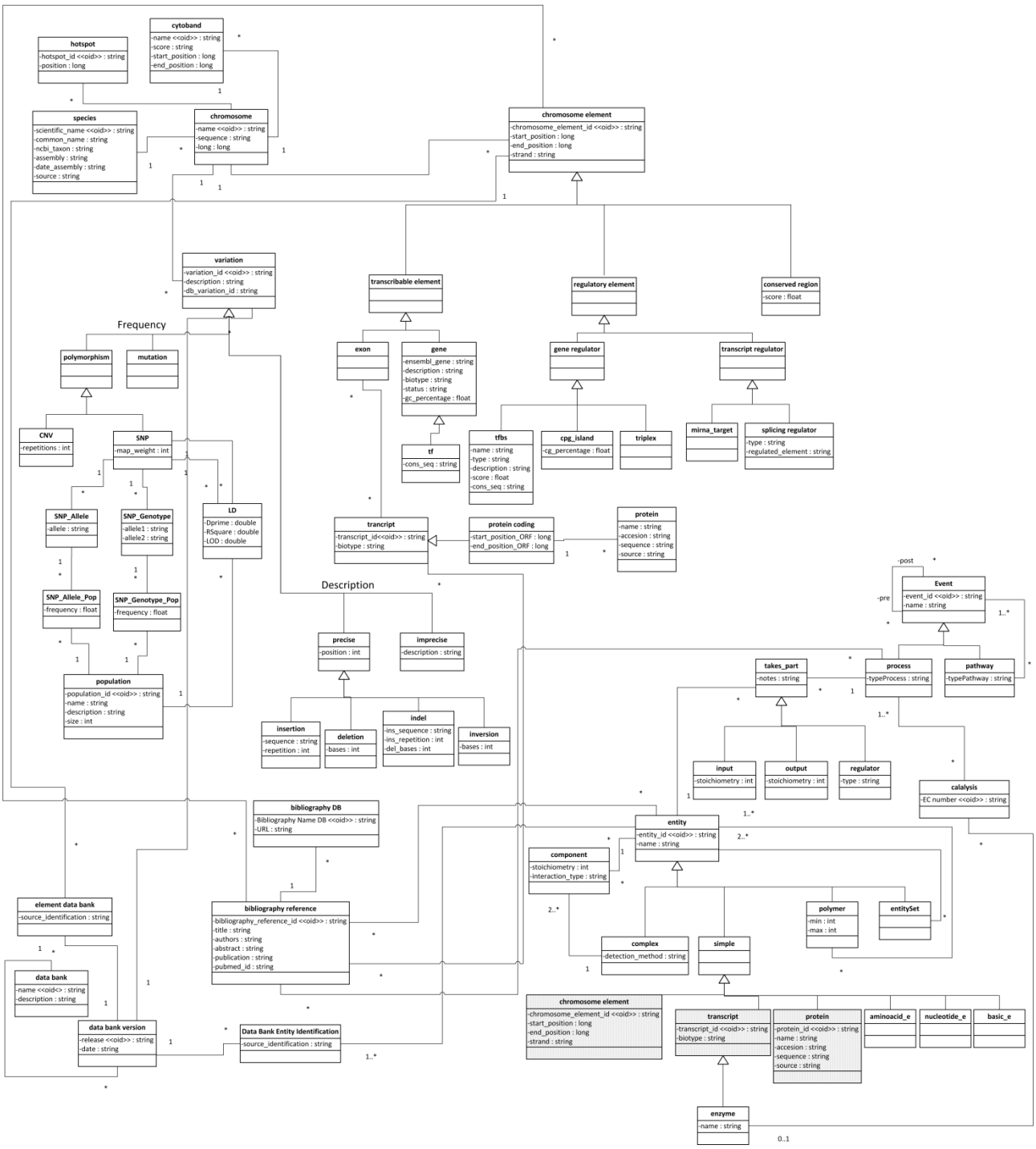


Ilustración 6-Modelado conceptual

3.1 Vista estructural

Esta vista (Ilustración 7), como su nombre indica, describe la estructura del genoma. La información genómica en un organismo se distribuye en 23 pares de cromosomas y genes que codifican proteínas, secuencias reguladores, etc. Por otro lado, cabe destacar que cada cromosoma pertenece a una única especie y que además contempla zonas calientes o hotspots [12] y subregiones llamadas citobandas que se hacen visibles microscópicamente después del tintado. A continuación se describen cada una de las clases que forman esta vista:

3.1.1 Chromosome

Clase	*Chromosome*
Descripción	*Chromosome* es la clase principal de esta vista, y se define como una estructura organizada y única dentro del ADN donde genes, elementos reguladores y otras secuencias de nucleótidos son localizados. Además, un cromosoma tiene una serie de atributos por los cuales es identificado
Atributos	
name	Nombre e identificador del cromosoma en la fuente de datos de la que se ha extraído la secuencia
Sequence	Secuencia de referencia del cromosoma
long	Campo longitud que indica el número de nucleótidos que tiene la secuencia.

Tabla 1-Chromosome

Hay que tener en cuenta, que debido a la cantidad de información genómica existente que va a ser almacenada en la posterior base de datos implementada, diferentes versiones del mismo genoma deben ser almacenadas en distintas versiones de la bases de datos y que además la secuencia de referencia almacenada no corresponde a ningún individuo en particular sino que se obtiene de una de las principales organizaciones de secuencias genómicas actuales.

3.1.2 Specie

Clase	*Specie*
Descripción	Como se ha comentado anteriormente, esta nueva versión del modelo conceptual no tiene cabida únicamente la especie humana, sino que abarca todas las especies conocidas en la actualidad. Por lo tanto la clase *specie*, sirve para determinar a qué familia pertenece cada uno de los cromosomas
Atributos	
scientific_name	Nombre científico e identificador por el cual se conoce la especie por ejemplo, homo sapiens.
common_name	Nombre común por el cual se conoce la especie. Por seguir con la analogía anterior el ejemplo aquí sería ser humano

ncbi_taxon_id	Identificador dado a una especie por la organización de NCBI
assembly	identificador de la versión utilizada como secuencia genómica de referencia de dicha especie
date_assembly	Fecha de la versión utilizada como secuencia genómica de referencia de dicha especie
source	Fuente de la cual se obtiene la secuencia genómica de referencia

Tabla 2-Specie

3.1.3 Hotspot

Clase	*hotspot*
Descripción	La clase *hotspot* describe otra característica del cromosoma representando información sobre los puntos en la secuencia de ADN donde existe mayor probabilidad de que se produzca la de recombinación durante el proceso de meiosis
Atributos	
hotspot_id	Identificador interno del cruce de recombinación.
position	Punto dentro de la secuencia de ADN en la que se produce el proceso de recombinación.

Tabla 3-Hotspot

3.1.4 Cytoband

Clase	*Cytoband*
Descripción	La clase *cytoband*, conocida también con el nombre de banda citogenética, describe también otra característica del cromosoma representando información sobre las subregiones de un cromosoma que llegan a ser visibles microscópicamente después del tintado durante una fase específica del ciclo celular. Una citobanda es representada mediante
Atributos	
name	el atributo nombre de la citobanda sigue siempre el mismo formato siguiendo las reglas establecidas que consisten en una “q” o una “p”, dependiendo del brazo del cromosoma, seguida de uno, dos o tres números separados por puntos dependiendo de la resolución utilizada i.e. (q24.22).
score	indica la intensidad de tintado, la cual puede tomar cinco valores diferentes proporcionales a la presencia de A y T.
start_position	posición inicial en la secuencia de referencia del cromosoma
end_position	posición final en la secuencia de referencia del cromosoma

Tabla 4-Cytoband

3.1.5 Chromosome element

Clase	*Chromosome Element*
Descripción	La clase *chromosome element* representa información sobre fragmentos relevantes dentro del cromosoma. Tiene cuatro atributos
Atributos	
chromosome_element_id	Odentificador interno de cada uno de los elementos del cromosoma.
start_position	Posición inicial del elemento en la secuencia de referencia del cromosoma.
end_position	Posición final del elemento en la secuencia de referencia del cromosoma.
strand	Hebra dentro de la doble hélice en la que se encuentra el elemento dentro del cromosoma.

Tabla 5-Chromosome element

Los elementos del cromosoma pueden ser de tres tipos dependiendo de la función que desempeñen: transcribable element, regulatory element and conserved region.

3.1.6 Transcribable element

Clase	*transcribable element*
Descripción	La clase *transcribable element* representa una región del ADN que se puede transcribir, o en otras palabras un elemento del que se crea un ARN complementario a partir de la secuencia de ADN. Este tipo de regiones pueden especializarse en dos tipos: gene y exon.

Tabla 6-Transcribable element

3.1.7 Gene

Clase	*gene*
Descripción	La clase *gene* representa una región de ADN que contiene la información necesaria para la síntesis de una macromolécula con una función celular específica, es decir contiene elementos reguladores que controlan el proceso de transcripción, normalmente sintetiza proteínas, pero también otro tipo de ARNs.
Atributos	
ensemble_gene	Nombre del gen proporcionado por el repositorio genómico Ensembl.
description:	Descripción del gene al que se hace referencia.
biotype	Especialización del tipo de gen dependiendo de las funciones que realiza, puede tomar valores como por ejemplo: snRNA,

	miRNA, protein coding, etc.
status	Determina el estado de validez en el que se encuentra cada elemento en la actualidad, puede tomar valores como: obsoleto, nuevo, etc.
gc_percentage	A diferencia del resto de regiones de la secuencia de ADN, ha sido comprobado que las regiones transcribibles tienen mayor alto contenido de Gs y Cs en su secuencia y que dicho contenido es directamente proporcional a la longitud de la secuencia codificante. Este atributo almacena el porcentaje de pares de bases Cs y Gs que existen en el elemento.

Tabla 7-Gene

Un gen, además, dependiendo del valor de su atributo biotype puede especializarse en diversos tipos de genes, dependiendo como se ha dicho anteriormente de las función que desempeñe. Existen muchos tipos de genes que podrían ser modelados, pero por simplificar el modelo se decide ilustrar un solo ejemplo, los factores de transcripción que se describen a continuación.

3.1.8 Tf (factor de transcripción)

Clase	*tf*
Descripción	La clase *tf* (factor de transcripción) representa aquellos genes que codifican una proteína cuya función es regular la transcripción de otros genes o incluyo la suya propia y se define mediante el atributo
Atributos	
cons_seq	Este atributo hace referencia a la secuencia de nucleótidos que una vez acoplada a las regiones de unión de la cadena de ADN realizará una función reguladora para el gen.
Sequence	Secuencia de referencia del cromosoma
long	Campo longitud que indica el número de nucleótidos que tiene la secuencia.

Tabla 8-TF

3.1.9 Exon

Clase	*Exon*
Descripción	La clase *exon* representa un elemento transcribible que forma parte del gen, y que es además la unidad básica de los transcritos. Cada exón codifica una porción específica de la proteína completa, de manera que el conjunto de exones forma la región codificante del gen.

Tabla 9-Exon

3.1.10 Regulatory element

Clase	*Regulatory element*
Descripción	La clase *regulatory* element representa regiones del ADN que realizan una función reguladora controlando ciertos procesos existentes dentro el ADN. Los elementos reguladores se especializan en dos clases dependiendo de si es un elemento regulador del gen o del transcrito: gene regulator y transcript regulator.

Tabla 10-Regulatory element

3.1.11 Gene regulator

Clase	*Gene regulator*
Descripción	La clase *gene regulator* representa los elementos reguladores del gen, entre los cuales se encuentran: tfbs, cpg_island y triplex

Tabla 11-Gene regulator

3.1.12 Tfbs (transcription factor binding sites)

Clase	*tfbs*
Descripción	La clase *tfbs* son regiones de unión de los factores de transcripción que producen un efecto en la transcripción del gen bien sea de activación o represión
Atributos	
name	Nombre que toma el sitio de unión de los factores de transcripción
type	Los sitios de unión de los factores de transcripción pueden ser de dos tipos dependiendo de la función que desempeñen: activador o inhibidor
description	Descripción del tfbs
score	Grado de similitud entre la secuencia consenso y el tfbs
cons_seq	Secuencia consenso la cual enlaza el tfbs

Tabla 12-TFBS

3.1.13 Cpg island

Clase	*cpg island*
Descripción	Las *cpg island* conforman aproximadamente un 40% de promotores de los genes de mamíferos. Son regiones donde existe una gran concentración de pares de Cs y Gs enlazados por fosfatos. La "p" en CpG representa que están enlazados por un fosfato y simboliza un conjunto de repeticiones de las bases CG que están cerca del promotor y son objetivos para la metilación que es otra manera de alterar la expresión del gen. La definición formal de una isla CpG es una región con al menos 200 pares de bases, con un porcentaje de GC mayor de 50 y con un promedio de CpG observado/esperado mayor de 0,6
Atributos	
cg_percentage	Representa el porcentaje de GC en el elemento

Tabla 13-CPG island

3.1.14 Triplex

Clase	*triplex*
Descripción	Los *triplex* son secuencias de ADN que se intercalan en la doble hélice de ADN de las células, pasando a tener éste tres cadenas, de tal manera que se impide el proceso de transcripción causando un efecto negativo en el individuo

Tabla 14-Triplex

3.1.15 Transcript regulator

Clase	*transcript regulator*
Descripción	La clase *transcript regulator* representa regiones reguladoras del transcrito. Existen muchas especializaciones de elementos reguladores del transcrito, pero por razones de simplificación en este modelo se representan únicamente dos: mirna target y splicing regulator.

Tabla 15-Transcript regulator

3.1.16 Mirna target

Clase	*Mirna target*
Descripción	La clase *Mirna target* representa una región reguladora del transcrito a la que se unirá post-transcripcionalmente un miRNA

Tabla 16-Mirna Target

3.1.17 Splicing regulator

Clase	*splicing regulator*
Descripción	La clase *splicing regulator* representa un elemento regulador del transcrito que regula el proceso de splicing
Atributos	
promover	Indica el tipo de regulación y puede tomar dos valores, desactivar (silencer) o promover (enhancer).
regulated_element	Indica cual es el elemento regulado si se trata de un intrón o un exón.

Tabla 17-Splicing regulator

3.1.18 Conserved región

Clase	*conserved region*
Descripción	La clase *conserved region* representa las regiones conservadas dentro del cromosoma, regiones que normalmente tienden a ser no codificantes, es decir, se mantienen intactas tras el proceso de evolución entre las especies.
Atributos	
score	Representa el grado de conservación de la región (puede tomar dos

	valores: o un valor estadístico indicando la probabilidad o un valor extraído de una fórmula).

Tabla 18-Conserved región

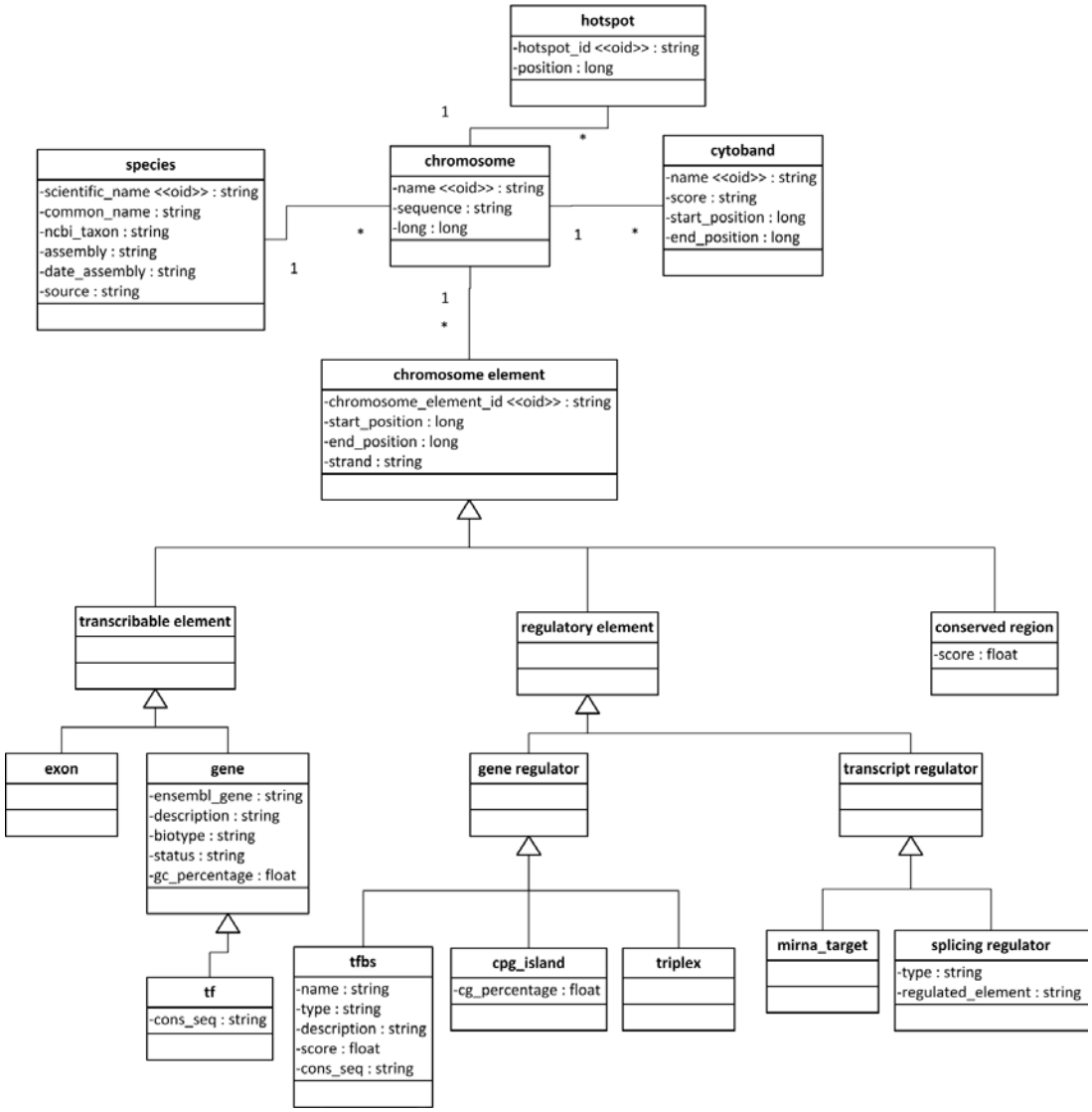


Ilustración 7-Vista estructural

3.2 Vista de transcripción

Un gran número de genes expresan su funcionalidad a través de la producción de proteínas. La vista transcripción representa los componentes y conceptos relacionados con la síntesis de proteínas.

La secuencia de ADN que se transcribe en una molécula de ARN codifica al menos un gen, y si el gen transcrito codifica para una proteína, el resultado de la transcripción es RNA mensajero (mRNA), el cual será entonces usado para crear esa proteína a través de un proceso de traducción.

Después de la transcripción, tiene lugar una modificación en el ARN llamada splicing, en la cual los intrones son borrados y los exones se unen. Pero en muchos de los casos, el proceso de splicing no es "perfecto" y puede variar la composición de los exones del mismo ARN mensajero. Este fenómeno es entonces llamado splicing alternativo. El splicing alternativo puede ocurrir de muchas maneras. Los exones pueden ser extendidos o saltados, o los intrones pueden ser retenidos.

A continuación se describen las clases que forman la vista.

3.2.1 Transcript

Clase	*transcript*
Descripción	La clase *transcript* representa los diferentes transcritos que presenta un gen. Estos transcritos están formados por una serie de exones. Como se ha comentado antes, existe un fenómeno llamado splicing alternativo que permite la combinación de diferentes exones, e incluso en poca medida algún intron, formando diferentes transcritos.
Atributos	
transcript	Identificador interno del transcrito.
biotype	Cada transcrito puede tener una función diferente representada con este atributo, que puede tomar el valor de: protein coding, trna, rrna, mirna, sirna, pirna, antisense, long noncoding, riboswitch, shrna, snorna, mitocondrial o otros

Tabla 19-Transcript

3.2.2 Protein coding

Clase	*protein coding*
Descripción	La clase *protein coding* es una especialización de la clase transcrito y que, como su propio nombre indica, representa el primero de los biotipos citados arriba. Ya que este tipo de transcritos sintetiza para una proteína se le añaden nuevos atributos
Atributos	
start_position_ORF	Este atributo hace referencia a la secuencia de nucleótidos que una vez acoplada a las regiones de unión de la cadena de ADN realizará una

	función reguladora para el gen.
end_position_ORF	Secuencia de referencia del cromosoma

Tabla 20-Protein coding

3.2.3 Protein

Clase	*protein*
Descripción	La clase *protein* da soporte a las miles de proteínas que se sintetizan a partir de un transcrito
Atributos	
name	nombre e identificador de la proteína
accesion	identificador que presenta la proteína en la fuente de datos de la cual ha sido extraída
sequence	la secuencia de la proteína
source	fuente de datos de la cual se ha extraído la información

Tabla 21-Protein

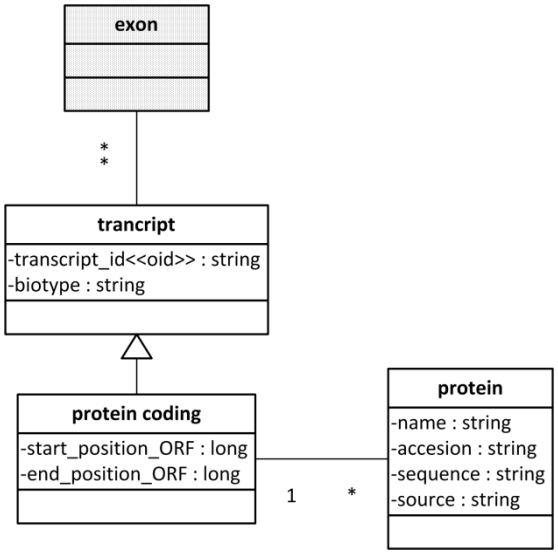


Ilustración 8-Vista de transcripción

3.3 Vista de variaciones

La vista variación modela el conocimiento relacionado con las diferencias encontradas en la secuencia de ADN de diversos individuos. A continuación se detallan las clases que la forman y la explicación de cada una de ellas:

3.3.1 Variation

Clase	*variation*
Descripción	La clase *variation* es la clase principal en esta vista, en ella se representan como su propio nombre indica, todas las variaciones existentes en la cadena de ADN
Atributos	
variation_id	identificador interno de la variación
description	proporciona una descripción de la variación de referencia del cromosoma
id_variation_db	identificador que proporciona la fuente de datos de la cual se ha extraído la variación.

Tabla 22-Protein

Las variaciones se especializan siguiendo dos criterios: la precisión en su descripción (ISA description) y su frecuencia (ISA frequency).

En la jerarquía frecuencia, o en otras palabras si la variación se presenta en más del 1% de la población o es un caso puntual, una variación puede estar especializada en dos clases: mutation y polimorphysm.

En la jerarquía descripción, una variación puede estar especializada en dos clases: precise e imprecise, dependiendo de si se conocen datos al respecto de su posición.

Por otra parte, cabe destacar que la clase Variation enlaza esta vista con la vista de la estructura del genoma, mediante una relación entre la clase Variation y la clase Chromosome_element que indica que una variación es un elemento que forma parte de un cromosoma.

3.3.2 Mutation

Clase	*mutation*
Descripción	La clase *mutation*, especialización de tipo ISA frequency, hace referencia a las variaciones con efecto patológico que se encuentran en un bajo porcentaje de la población, es decir, en menos del 1%.

Tabla 23-Mutation

3.3.3 Polymorphism

Clase	*polymorphism*
Descripción	La clase *polymorphism*, especialización de tipo ISA frequency, describe las variaciones que aparecen en más del 1% de la población y normalmente no tienen un diagnostico maligno, por lo que se heredan de generación en generación. Este tipo de variaciones, puede especializarse en dos tipos: CNV (Copy Number Variation) y SNP (Single Nucleotide Polymorphism).

Tabla 24-Polymorphism

3.3.4 CNV

Clase	*cnv*
Descripción	Un *cnv* (copy number variation) es definido como una variación que consiste en la repetición un cierto número de veces o el borrado de una pequeña región de la secuencia de ADN
Atributos	
repetitions	almacena el número de veces que la secuencia se repite o se borra

Tabla 25-CNV

3.3.5 SNP

Clase	*SNP*
Descripción	Un *SNP* es un polimorfismo que tiene lugar cuando un único nucleótido dentro del genoma difiere de lo habitual entre individuos de la misma especie agrupados por poblaciones. Estas variaciones en la secuencia del ADN pueden afectar a la respuesta de los individuos a enfermedades, bacterias, virus, productos químicos, fármacos, etc
Atributos	
map_weight	las veces que dicho SNP ha sido mapeado en la muestra del genoma de un individuo.

Tabla 26-SNP

Un SNP es un cambio de un único nucleótido en una posición del genoma pero a su vez puede proporcionar datos relevantes: los distintos valores que puede tomar el SNP teniendo en cuenta un único alelo (SNP_Allele) y las diferentes combinaciones de valores que puede tomar el SNP teniendo en cuenta los dos alelos (SNP_Genotype). Además, existe más información de interés con respecto a los SNPs así como el linkage disequilibrium (LD) y que se describe como marcador que indica la relación existente entre dos SNPs dentro de una población.

3.3.6 SNP_Allele

Clase	*SNP_Allele*
Descripción	La clase *SNP_Allele* representa los diferentes valores que puede tomar un SNP teniendo en cuenta un solo alelo
Atributos	
allele	este atributo indica el valor que puede tomar el alelo en cada caso. Su dominio es {A,T,G,C}.

Tabla 27-SNP_Allele

3.3.7 SNP_Genotype

Clase	*SNP_Genotype*
Descripción	La clase *SNP_Genotype* representa los diferentes valores que pueden tomar el par de alelos de cada individuo en la posición del SNP teniendo en cuenta las dos hebras
Atributos	
allele1	este atributo indica el valor que puede tomar el alelo en una hebra. Su dominio es {A,T,G,C}.
allele2	este atributo indica el valor que puede tomar el alelo en la otra ebra. Su dominio es {A,T,G,C}.

Tabla 28-SNP_Genotype

Como se ha comentado en la descripción de SNP, cada uno de ellos está directamente relacionado con varias poblaciones, por lo que las dos clases, SNP_Allele y SNP_Genotype tienen relación con varias poblaciones en cada caso. Para proporcionar información sobre la frecuencia de aparición de cada SNP en diferentes poblaciones, bien sea a nivel alélico o a nivel genotípico, se crean también las clases SNP_Allele_Pop y SNP_Genotype_Pop.

3.3.8 SNP_Allele_Pop

Clase	*SNP_Allele_Pop*
Descripción	La clase *SNP_Allele_Pop* representa la frecuencia en la que cada SNP, teniendo únicamente en cuenta un alelo, aparece en cada población
Atributos	
frequency _seq	frecuencia con la que cada SNP aparece en diversas poblaciones

Tabla 29-SNP_Allele_Pop

3.3.9 SNP_Genotype_Pop

Clase	*SNP_Genotype_Pop*
Descripción	La clase *SNP_ Genotype _Pop* representa la frecuencia en la que cada SNP aparece en cada población teniendo únicamente en cuenta los dos alelos
Atributos	
frequency	frecuencia con la que cada SNP aparece en diversas poblaciones.

Tabla 30-SNP_Genotype_Pop

3.3.10 Population

Clase	*population*
Descripción	La clase *population* representa conjuntos de individuos con características comunes
Atributos	
name	nombre e identificador de cada población.
description	descripción de cada población
size	cantidad de individuos pertenecientes a una población

Tabla 31-Population

3.3.11 LD

Clase	*LD*
Descripción	Otro concepto modelado que hemos nombrado anteriormente es el linkage disequilibrium o *LD* que es un marcador que define la relación existente entre dos SNPs en una población específica
Atributos	
Dprime, Rsquare y LOD	los tres son valores matemáticos de ámbito muy biológico a los que no vamos a entrar en detalle en esta tesis

Tabla 32-LD

3.3.12 Precise

Clase	*precise*
Descripción	La clase *precise*, especialización del tipo ISA Description, representa las variaciones detectadas con posición conocida dentro del cromosoma en la secuencia de ADN
Atributos	
cons_ position	posición en la que se encuentra la variación dentro de la secuencia del cromosoma.

Tabla 33-Precise

La clase Precise se especializa en cuatro nuevas entidades dependiendo de qué tipo de variación haya tenido lugar dentro del genoma: insertion, deletion, indel e inversion.

3.3.13 Insertion

Clase	*insertion*
Descripción	La clase *insertion* representa variaciones que consisten en la inserción de una secuencia de nucleótidos un número de veces en la secuencia de ADN del cromosoma
Atributos	
sequence	secuencia de nucleótidos insertados en la secuencia
repetition	número de veces que se repite la secuencia insertada

Tabla 34-Insertion

3.3.14 Deletion

Clase	*deletion*
Descripción	La clase *deletion* representa variaciones que consisten en el borrado de un número de nucleótidos en la secuencia de ADN del cromosoma
Atributos	
bases	número de nucleótidos borrados en la secuencia.

Tabla 35-Deletion

3.3.15 Indel

Clase	*indel*
Descripción	La clase *indel* representa variaciones consistentes en inserciones y borrados a la vez en la secuencia de ADN del cromosoma
Atributos	
ins_sequence	secuencia de nucleótidos insertados en la secuencia
ins_repetition	número de veces que se repite la secuencia insertada
del_bases	número de nucleótidos borrados

Tabla 36-Indel

3.3.16 Inversion

Clase	*inversion*
Descripción	La clase *inversion* representa variaciones que invierten el orden de una secuencia de nucleótidos en la secuencia del cromosoma.
Atributos	
bases	número de nucleótidos invertidos en la secuencia

Tabla 37-Inversion

3.3.17 Imprecise

Clase	*imprecise*
Descripción	La clase *imprecise* dentro de la jerarquía de descripción representa variaciones cuya posición es desconocida dentro de la secuencia de ADN.
Atributos	
description	descripción de la variación en lenguaje natural

Tabla 38-Imprecise

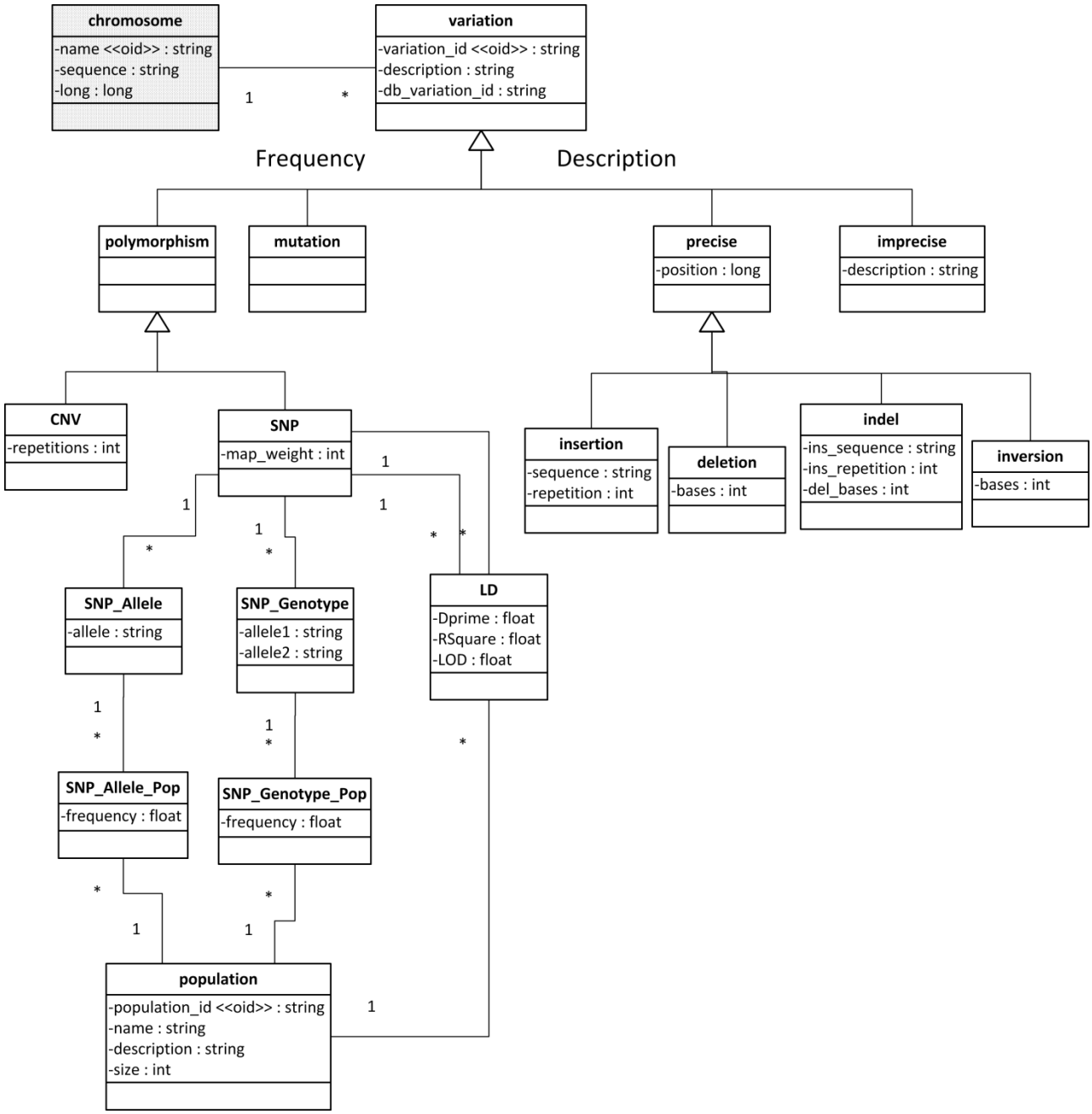


Ilustración 9-Vista de variaciones

3.4 Vista de rutas metabólicas

En bioquímica, las rutas metabólicas (pathways) (Ilustración 10), son una serie de reacciones químicas que ocurren dentro de una célula [13]. Esta composición de procesos viene representada en el esquema por las siguientes clases:

3.4.1 Event

Clase	*event*
Descripción	La clase *event* es la clase principal y es la que representa la combinación de procesos existentes en el organismo
Atributos	
event_id	identificador interno del evento
name	nombre que tiene el evento.

Tabla 39-Event

Además, la clase Event se especializa en dos clases dependiendo de la cantidad de procesos que lo formen: Process y Pathway.

3.4.2 Process

Clase	*process*
Descripción	La clase *process* representa un único proceso atómico o dicho en otras palabras un proceso de tipo simple.

Tabla 40-Process

3.4.3 Pathway

Clase	*pathway*
Descripción	La clase *pathway* representa un proceso complejo formado por una secuencia de otros procesos de tipo complejo o simple

Tabla 41-Pathway

Este conocimiento es modelado con las siguientes clases: takes_part, input, output y regulator.

3.4.4 Takes_part

Clase	*takes_part*
Descripción	La clase *takes_part* es una clase genérica que define de que manera una entidad participa dentro de uno o varios procesos
Atributos	
notes	comentario sobre la relación entre las entidades que toman parte en cada proceso.

Tabla 42-Takes_part

Se especializa en tres entidades diferentes dependiendo de la manera en la que dicha entidad participe en dicho proceso: input, output y regulator.

3.4.5 Input

Clase	*input*
Descripción	La clase *input* representa la entidad de entrada a un proceso
Atributos	
stoichiometry	cantidad de la entidad que interviene en el proceso

Tabla 43-Input

3.4.6 Output

Clase	*output*
Descripción	La clase *output* representa el resultado final del proceso

Tabla 44-Output

3.4.7 Regulator

Clase	*regulator*
Descripción	La clase *regulator* como su propio nombre indica los procesos reguladores existentes en las partes intermedias de la reacción,
Atributos	
type	se usa para distinguir de qué tipo de regulación se trata y puede tomar dos valores: inhibidor y activador

Tabla 45-Regulator

3.4.8 Catalysis

Clase	*catalysis*
Descripción	La clase *catalysis*, define el proceso por el cual se aumenta o disminuye la velocidad de una reacción química. Es un tipo especial de regulador de pathways que ha sido modelada aparte debido al hecho de que se tiene constancia de que forma parte de muchos procesos pero en algunos de ellos el catalizador es desconocido. En los casos en los que el catalizador es conocido, una enzima es asociada al correspondiente proceso
Atributos	
EC number	los números EC (Enzyme Commission numbers) son un esquema de clasificación numérica para las enzimas, basado en las reacciones químicas que catalizan. En realidad los números EC codifican reacciones catalizadas por enzimas. Enzimas diferentes (por ejemplo que procedan de organismos diferentes) que catalicen la misma reacción recibirán el mismo número EC. Cada código de enzimas consiste en las dos letras EC seguidas por 4 números separados por puntos. Estos números representan una clasificación progresivamente más específica. Por ejemplo, la enzima tripéptido aminopeptidasa tiene el código EC 3.4.11.4

Tabla 46-Catalysis

3.4.9 Enzime

Clase	*enzime*
Descripción	La clase *enzime*, es una especialización de proteína que cataliza reacciones químicas. Una enzima hace que una reacción química que es energéticamente posible pero que transcurre a una velocidad muy baja, sea cinéticamente favorable, es decir, transcurra a mayor velocidad que sin la presencia de la enzima. Está asociada con el proceso de catálisis para determinar cuál es el catalizador en caso de ser conocido
Atributos	
name	las enzimas son usualmente nombradas de acuerdo a la reacción que producen. Normalmente, el sufijo "-asa" es agregado al nombre del sustrato (p. ej., la lactasa es la enzima que degrada lactosa) o al tipo de reacción (p. ej., la ADN polimerasa forma polímeros de ADN).

Tabla 47-Enzime

3.4.10 Entity

Clase	*entity*
Descripción	La clase *entity* es la clase genérica que representa el tipo de entidades que pueden participar en un proceso de un pathway
Atributos	
entity_id	identificador interno de la clase entity

name	atributo genérico que proporciona información acerca del nombre de la entidad.

Tabla 48-Entity

3.4.11 Complex

Clase	*complex*
Descripción	La clase *complex* representa entidades que están formadas por la combinación de otras entidades más simples
Atributos	
detection_method	este atributo indica la técnica usada para determinar cómo se ha formado la entidad.

Tabla 49-Complex

3.4.12 Component

Clase	*component*
Descripción	La clase *component* representa de que manera una entidad complex está formada por sus entidades más simples
Atributos	
stoichiometry	permite conocer cuanta cantidad del complejo está formado por cada uno de sus componentes.
interaction	permite conocer como el complejo ha sido formado a partir de cada uno de sus componentes

Tabla 50-Component

3.4.13 Polymer

Clase	*polymer*
Descripción	La clase *polymer* representa entidades que son generadas por la repetición de alguna entidad, bien sea compleja o simple
Atributos	
min	representa el rango de repeticiones mínimo de la entidad que forma el polímero.
max	representa el rango de repeticiones máximo de la entidad que forma el polímero

Tabla 51-Polymer

3.4.14 Simple

Clase	*simple*
Descripción	La clase *simple*, representa las entidades más simples que pueden formar parte de un proceso, como por ejemplo: gen, ARN, proteína,

	aminoácido, nucleótido, entidad básica (agua, fósforo, etc.).

Tabla 52-Simple

3.4.15 EntitySet

Clase	*entitySet*
Descripción	La clase *entitySet* representa un conjunto de entidades que participan de manera habitual conjuntamente en algunos procesos, lo que permite reducir la cantidad de procesos similares existentes

Tabla 53-EntitySet

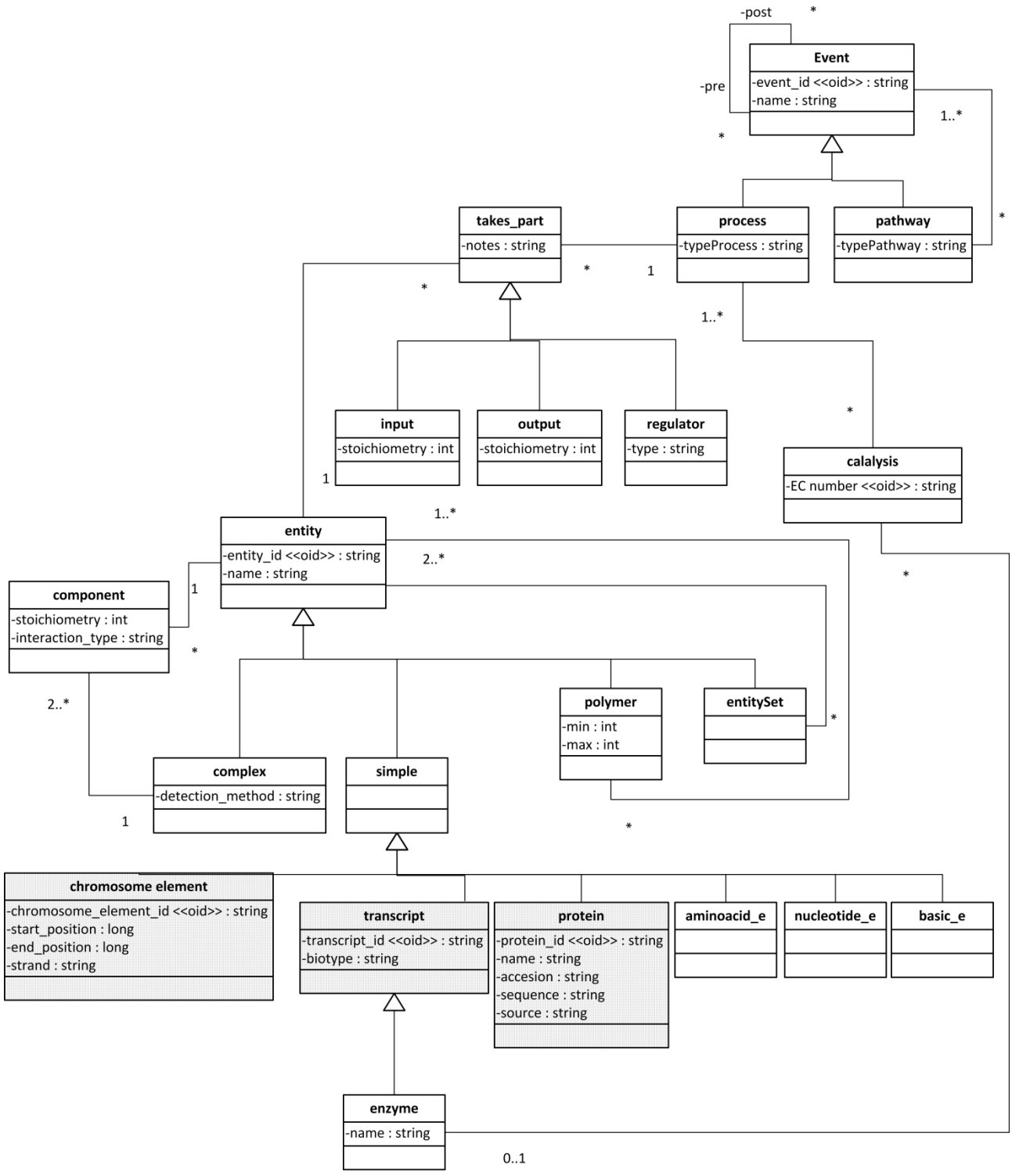


Ilustración 10- Vista de rutas metabólicas

3.5 Vista de fuente de datos y bibliografía

Esta vista proporciona información sobre las fuentes de datos de las que se ha extraído la información que se va a almacenar en el modelo, así como una serie de documentos bibliográficos de consulta para quien desee obtener más información con respecto a algún aspecto aquí definido.

Para mantener información sobre las fuentes de las cuales se ha obtenido la información, esta vista incluye las siguientes clases:

3.5.1 Data_bank

Clase	*data_bank*
Descripción	La clase *data_bank* proporciona información sobre la fuente de datos de la cual se extrae la información de cada uno de los elementos del modelo
Atributos	
name	nombre de la fuente de datos
description	descripción de la fuente de datos

Tabla 54-Data_bank

3.5.2 Data bank versión

Clase	*data_bank_version*
Descripción	Esta clase *data_bank_version* proporciona información sobre la versión de cada una de las bases de datos que se han utilizado y en qué fecha dichas bases de datos han sido actualizadas
Atributos	
release	versión de la fuente de datos
date	fecha en la que se actualizó por última vez la fuente de datos consultada

Tabla 55-Data_bank_version

3.5.3 Element data bank

Clase	*element data bank*
Descripción	La clase *element data bank* permite relacionar cada uno de los elementos del cromosoma de que fuente de datos han sido extraídos y su versión
Atributos	
source_identification	este atributo indica el identificador que proporciona cada una de las fuentes a los elementos del cromosoma

Tabla 56-Element_data_bank

3.5.4 Data Bank Entity Identification

Clase	Data Bank Entity Identification
Descripción	Permite relacionar cada una de las entidades que forman los pathways con la fuente de datos y la versión de la cual se ha extraído la información
Atributos	
source_identification	este atributo indica el identificador que proporciona cada una de las fuentes a las entidades que forman los pathways

Tabla 57-Data_bank_Entity_Identificaction

3.5.5 Bibliography DB

Clase	*tf*
Descripción	La clase *bibliography DB* representa las distintas fuentes de datos de la web de las que se extraen las publicaciones científicas
Atributos	
Bibliography Name DB	nombre de la base de datos de la que se extraen las publicaciones científicas.
URL	dirección web de la base de datos de las que se extraen las publicaciones.

Tabla 58-Bibliography_DB

3.5.6 Bibliography reference

Clase	*Bibliography reference*
Descripción	*Bibliography reference* proporciona información sobre los artículos relacionados con cada uno de los elementos almacenados si se dispone de ella
Atributos	
bibliography_reference_id	identificador interno de las referencias bibliográficas
title	título del artículo
authors	autores que han escrito el artículo
abstract	resumen del artículo
Publication	fecha en la cual se ha publicado el artículo
pubmed_id	identificador que la base de datos de pubmed proporciona al artículo

Tabla 59-Bibliography_reference

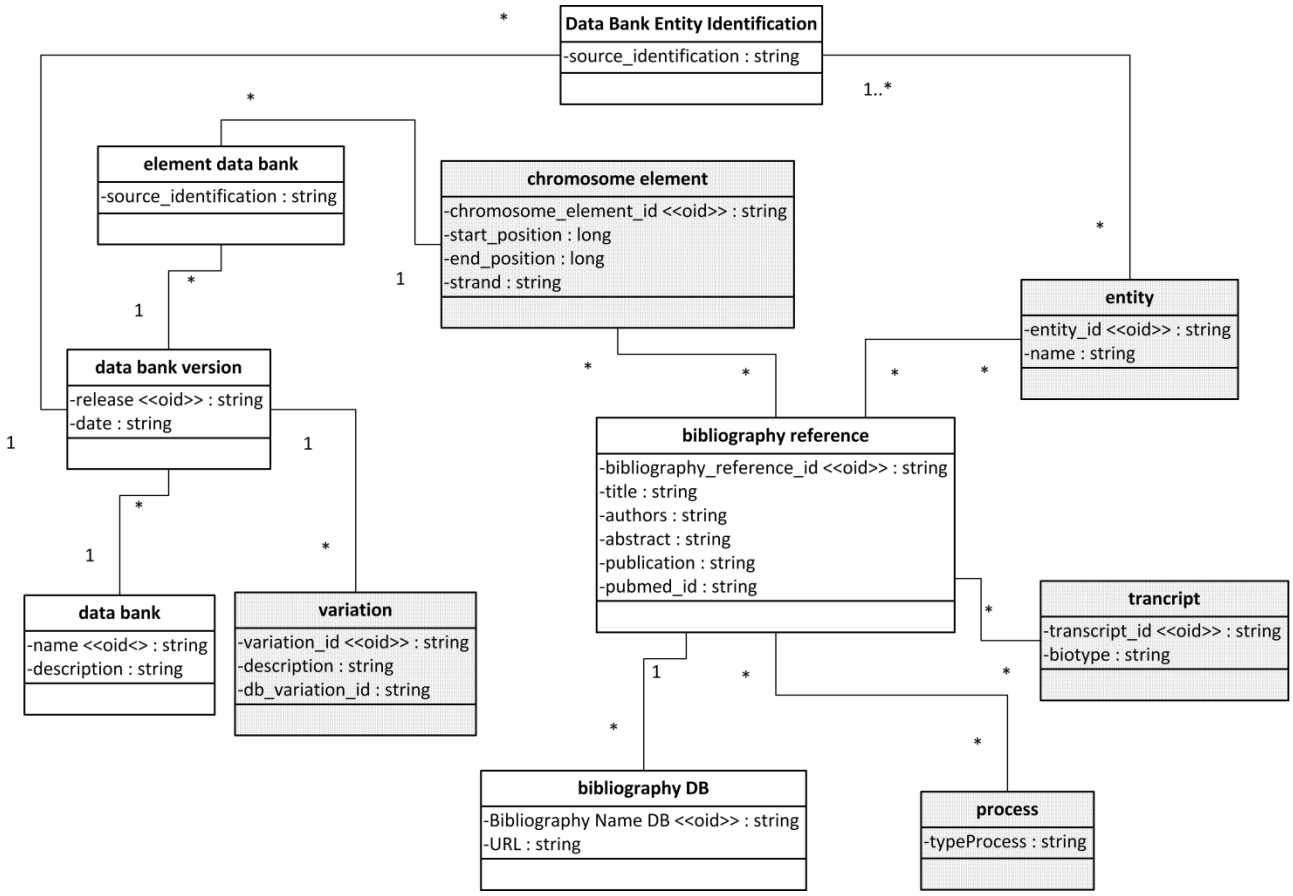


Ilustración 11-Vista fuentes de datos y bibliografía del modelo conceptual

3.6 Esquema relacional

El modelo relacional para la gestión de una base de datos es un modelo de datos basado en la lógica de predicados y en la teoría de conjuntos [14]. Este modelo es utilizado en la actualidad por analistas para modelar problemas reales y administrar datos dinámicamente.

Una vez modelado el sistema de información genómico el siguiente paso a realizar es la carga e integración de los diferentes repositorios. Para ello es necesaria la generación de un modelo relacional a partir de nuestro modelo conceptual.

El modelo conceptual genómico representa aquella información relevante en el campo de la genómica. Dado que muchas clases no van a ser utilizadas, únicamente se generará un esquema relacional que contenga las clases relevantes para el estudio. Estas clases están estructuradas en las diferentes vistas: la Vista estructural, La vista de variaciones, Vista de fuentes de datos, Vista de usuarios y validaciones y Vista de bibliografía.

Estas vistas son acordes a las vistas realizadas en el modelado conceptual. Sin embargo aparece una nueva vista no contemplada en el modelo, la Vista de usuarios y validaciones. Esta

vista ofrece la posibilidad de recoger las diferentes validaciones realizadas sobre una variación por los diferentes usuarios. En nuestro caso de estudio esta vista nos es irrelevante.

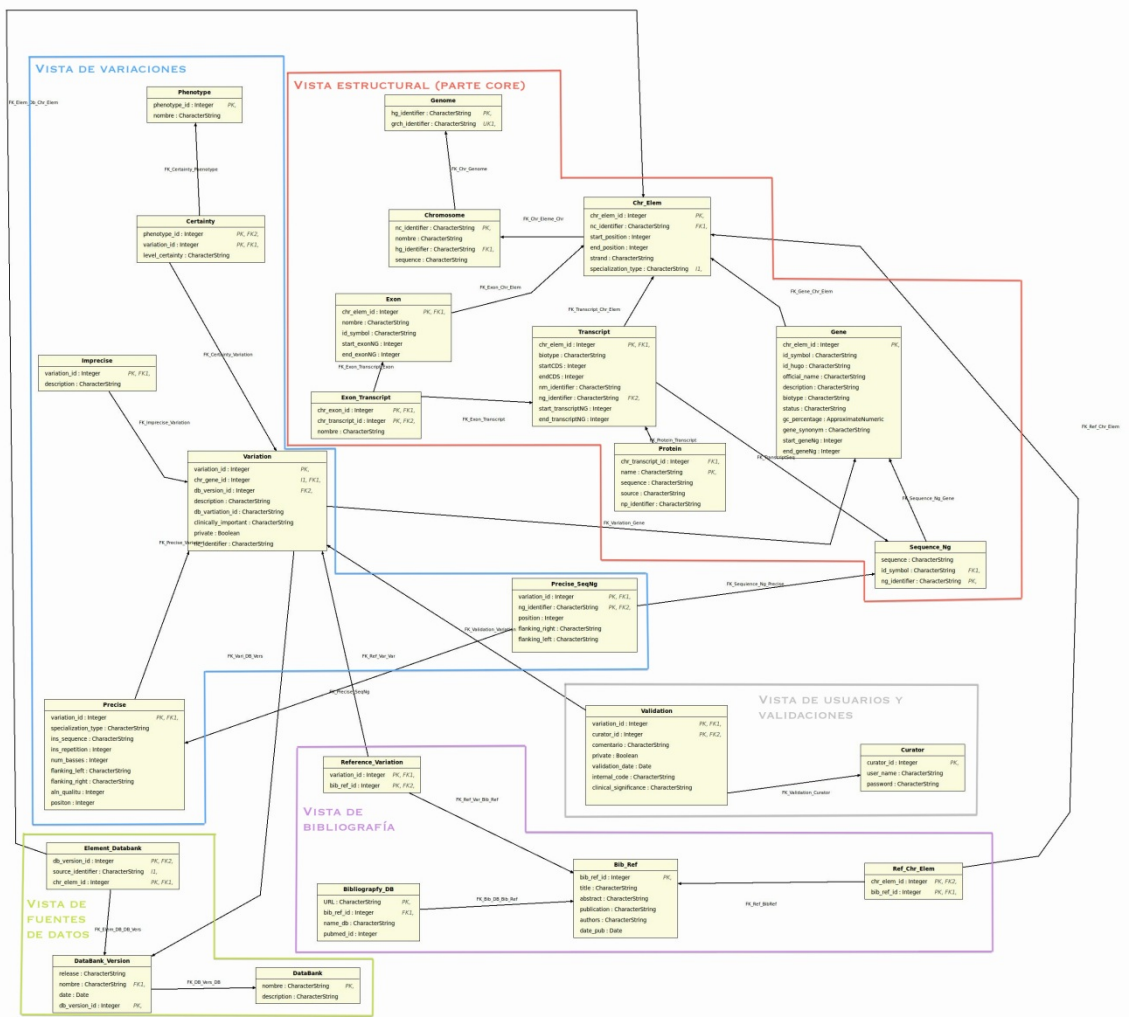


Ilustración 12-Esquema relacional

A partir de este esquema relacional se generan los scripts necesarios para la creación de la base de datos. Estos scripts dan lugar a las siguientes tablas en nuestro repositorio de información:

BIB_REF

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	BIB_REF_ID	NUMBER(38,0)	No	(null)	1	(null)
2	TITLE	VARCHAR2(4000 BYTE)	Yes	(null)	2	(null)
3	ABSTRACT	VARCHAR2(4000 BYTE)	Yes	(null)	3	(null)
4	PUBLICATION	VARCHAR2(4000 BYTE)	Yes	(null)	4	(null)
5	AUTHORS	VARCHAR2(4000 BYTE)	Yes	(null)	5	(null)
6	DATE_PUB	DATE	Yes	(null)	6	(null)

Ilustración 13-Tabla Bib_ref

BIBLIOGRAPHY_DB

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	URL	VARCHAR2(255 BYTE)	No	(null)	1	(null)
2	BIB_REF_ID	NUMBER(38,0)	Yes	(null)	2	(null)
3	NAME_DB	VARCHAR2(255 BYTE)	Yes	(null)	3	(null)
4	PUBMED_ID	NUMBER(38,0)	Yes	(null)	4	(null)

Ilustración 14-Tabla Bibliography_db

CERTAINTY

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PHENOTYPE_ID	NUMBER(38,0)	No	(null)	1	(null)
2	VARIATION_ID	NUMBER(38,0)	No	(null)	2	(null)
3	LEVEL_CERTAINTY	VARCHAR2(1 BYTE)	Yes	(null)	3	(null)

Ilustración 15-Tabla Certainty

CHR_ELEM

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CHR_ELEM_ID	NUMBER(38,0)	No	(null)	1	(null)
2	NC_IDENTIFIER	VARCHAR2(15 BYTE)	Yes	(null)	2	(null)
3	START_POSITION	NUMBER(38,0)	Yes	(null)	3	(null)
4	END_POSITION	NUMBER(38,0)	Yes	(null)	4	(null)
5	STRAND	VARCHAR2(2 BYTE)	Yes	(null)	5	(null)
6	SPECIALIZATION_TYPE	VARCHAR2(25 BYTE)	Yes	(null)	6	(null)

Ilustración 16-Tabla Chr_elem

CHROMOSOME

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	NC_IDENTIFIER	VARCHAR2(15 BYTE)	No	(null)	1	(null)
2	NOMBRE	VARCHAR2(5 BYTE)	Yes	(null)	2	(null)
3	HG_IDENTIFIER	VARCHAR2(9 BYTE)	Yes	(null)	3	(null)
4	SEQUENCE	CLOB	Yes	(null)	4	(null)

Ilustración 17-Tabla Chromosome

CURATOR

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CURATOR_ID	NUMBER(38,0)	No	(null)	1	(null)
2	USER_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2	(null)
3	PASS	VARCHAR2(30 BYTE)	Yes	(null)	3	(null)

Ilustración 18-Tabla Curator

DATABANK

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	NOMBRE	VARCHAR2(100 BYTE)	No	(null)	1	(null)
2	DESCRIPTION	VARCHAR2(255 BYTE)	Yes	(null)	2	(null)
3	URL	VARCHAR2(100 BYTE)	Yes	(null)	3	(null)

Ilustración 19-Tabla Databank

DATABANK_VERSION

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	RELEASE	VARCHAR2(255 BYTE)	Yes	(null)	1	(null)
2	NOMBRE	VARCHAR2(100 BYTE)	Yes	(null)	2	(null)
3	FECHA	DATE	Yes	(null)	3	(null)
4	DB_VERSION_ID	NUMBER(38,0)	No	(null)	4	(null)

Ilustración 20-Tabla Databank_version

ELEMENT_DATABANK

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	DB_VERSION_ID	NUMBER(38,0)	No	(null)	1	(null)
2	SOURCE_IDENTIFIER	VARCHAR2(20 BYTE)	Yes	(null)	2	(null)
3	CHR_ELEM_ID	NUMBER(38,0)	No	(null)	3	(null)

Ilustración 21-Tabla Element_databank

EXON

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CHR_ELEM_ID	NUMBER(38,0)	No	(null)	1	(null)
2	NOMBRE	VARCHAR2(20 BYTE)	Yes	(null)	2	(null)
3	ID_SYMBOL	VARCHAR2(10 BYTE)	Yes	(null)	3	(null)
4	START_EXONNG	NUMBER(38,0)	Yes	(null)	4	(null)
5	END_EXONNG	NUMBER(38,0)	Yes	(null)	5	(null)

Ilustración 22-Tabla Exon

EXON_TRANSCRIPT

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CHR_EXON_ID	NUMBER(38,0)	No	(null)	1	(null)
2	CHR_TRANSCRIPT_ID	NUMBER(38,0)	No	(null)	2	(null)
3	NOMBRE	VARCHAR2(30 BYTE)	Yes	(null)	3	(null)

Ilustración 23-Tabla Exon_transcript

GENE

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CHR_ELEM_ID	NUMBER(38,0)	No	(null)	1	(null)
2	ID_SYMBOL	VARCHAR2(10 BYTE)	Yes	(null)	2	(null)
3	ID_HUGO	VARCHAR2(10 BYTE)	Yes	(null)	3	(null)
4	OFFICIAL_NAME	VARCHAR2(100 BYTE)	Yes	(null)	4	(null)
5	DESCRIPTION	VARCHAR2(3000 BYTE)	Yes	(null)	5	(null)
6	BIOTYPE	VARCHAR2(30 BYTE)	Yes	(null)	6	(null)
7	STATUS	VARCHAR2(30 BYTE)	Yes	(null)	7	(null)
8	GC_PERCENTAGE	FLOAT	Yes	(null)	8	(null)
9	GENE_SYNONYM	VARCHAR2(250 BYTE)	Yes	(null)	9	(null)
10	START_GENENG	NUMBER(38,0)	Yes	(null)	10	(null)
11	END_GENENG	NUMBER(38,0)	Yes	(null)	11	(null)

Ilustración 24-Tabla Gene

GENOME

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	HG_IDENTIFIER	VARCHAR2(9 BYTE)	No	(null)	1	(null)
2	GRCH_IDENTIFIER	VARCHAR2(13 BYTE)	No	(null)	2	(null)

Ilustración 25-Tabla Genoma

IMPRECISE

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	VARIATION_ID	NUMBER(38,0)	No	(null)	1	(null)
2	DESCRIPTION	VARCHAR2(255 B...	Yes	(null)	2	(null)

Ilustración 26-Tabla Imprecise

PHENOTYPE_ID

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PHENOTYPE_ID	NUMBER(38,0)	No	(null)	1	(null)
2	NOMBRE	VARCHAR2(255 B...	Yes	(null)	2	(null)

Ilustración 27-Tabla Phenotype_id

PRECISE

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	VARIATION_ID	NUMBER(38,0)	No	(null)	1	(null)
2	SPECIALIZATION_TYPE	VARCHAR2(25 BYTE)	Yes	(null)	2	(null)
3	INS_SEQUENCE	VARCHAR2(1000 ...	Yes	(null)	3	(null)
4	INS_REPETITION	NUMBER(38,0)	Yes	(null)	4	(null)
5	NUM_BASES	NUMBER(38,0)	Yes	(null)	5	(null)
6	FLANKING_LEFT	VARCHAR2(60 BYTE)	Yes	(null)	6	(null)
7	FLANKING_RIGHT	VARCHAR2(60 BYTE)	Yes	(null)	7	(null)
8	ALN_QUALITY	NUMBER(38,0)	Yes	(null)	8	(null)
9	POSITION	NUMBER(38,0)	Yes	(null)	9	(null)

Ilustración 28-Tabla Precise

PRECISE_SEQNG

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	VARIATION_ID	NUMBER(38,0)	No	(null)	1	(null)
2	NG_IDENTIFIER	VARCHAR2(15 BYTE)	No	(null)	2	(null)
3	POSITION	NUMBER(38,0)	Yes	(null)	3	(null)
4	FLANKING_RIGHT	VARCHAR2(25 BYTE)	Yes	(null)	4	(null)
5	FLANKING_LEFT	VARCHAR2(25 BYTE)	Yes	(null)	5	(null)

Ilustración 29-Tabla Precise_seqng

PROTEIN

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CHR_TRANSCRIPT_ID	NUMBER(38,0)	Yes	(null)	1	(null)
2	NAME	VARCHAR2(100 BYTE)	No	(null)	2	(null)
3	SEQUENCE	CLOB	Yes	(null)	3	(null)
4	SOURCE	VARCHAR2(100 BYTE)	Yes	(null)	4	(null)
5	NP_IDENTIFIER	VARCHAR2(15 BYTE)	Yes	(null)	5	(null)

Ilustración 30-Tabla Protein

REF_CHR_ELEM

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CHR_ELEM_ID	NUMBER(38,0)	No	(null)	1	(null)
2	BIB_REF_ID	NUMBER(38,0)	No	(null)	2	(null)

Ilustración 31-Tabla Ref_chr_elem

REFERENCE_VARIATION

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	VARIATION_ID	NUMBER(38,0)	No	(null)	1	(null)
2	BIB_REF_ID	NUMBER(38,0)	No	(null)	2	(null)

Ilustración 32-Tabla Reference_variation

SEQUENCE_NG

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	SEQUENCE	CLOB	Yes	(null)	1	(null)
2	ID_SYMBOL	VARCHAR2(10 BYTE)	Yes	(null)	2	(null)
3	NG_IDENTIFIER	VARCHAR2(15 BYTE)	No	(null)	3	(null)

Ilustración 33-Tabla Sequence_ng

TRANSCRIPT

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CHR_ELEM_ID	NUMBER(38,0)	No	(null)	1	(null)
2	BIOTYPE	VARCHAR2(30 BYTE)	Yes	(null)	2	(null)
3	STARTCDS	NUMBER(38,0)	Yes	(null)	3	(null)
4	ENDCDS	NUMBER(38,0)	Yes	(null)	4	(null)
5	NM_IDENTIFIER	VARCHAR2(15 BYTE)	Yes	(null)	5	(null)
6	NG_IDENTIFIER	VARCHAR2(15 BYTE)	Yes	(null)	6	(null)
7	START_TRANSCRIPTNG	NUMBER(38,0)	Yes	(null)	7	(null)
8	END_TRANSCRIPTNG	NUMBER(38,0)	Yes	(null)	8	(null)

Ilustración 34-Tabla Transcript

VALIDATION

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	VARIATION_ID	NUMBER(38,0)	No	(null)	1	(null)
2	CURATOR_ID	NUMBER(38,0)	No	(null)	2	(null)
3	COMENTARIO	VARCHAR2(255 BYTE)	Yes	(null)	3	(null)
4	PRIVADO	NUMBER(38,0)	Yes	(null)	4	(null)
5	VALIDATION_DATE	DATE	Yes	(null)	5	(null)
6	INTERNAL_CODE	VARCHAR2(50 BYTE)	Yes	(null)	6	(null)
7	CLINICAL_SIGNIFICANCE	VARCHAR2(250 BYTE)	Yes	(null)	7	(null)

Ilustración 35-Tabla Validation

VARIATION

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	VARIATION_ID	NUMBER(38,0)	No	(null)	1	(null)
2	CHR_GENE_ID	NUMBER(38,0)	Yes	(null)	2	(null)
3	DB_VERSION_ID	NUMBER(38,0)	Yes	(null)	3	(null)
4	DESCRIPTION	VARCHAR2(1000 BYTE)	Yes	(null)	4	(null)
5	DB_VARIATION_ID	VARCHAR2(50 BYTE)	Yes	(null)	5	(null)
6	CLINICALLY_IMPORTANT	VARCHAR2(50 BYTE)	Yes	(null)	6	(null)
7	PRIVADO	NUMBER(38,0)	Yes	(null)	7	(null)
8	NC_IDENTIFIER	VARCHAR2(15 BYTE)	Yes	(null)	8	(null)
9	NG_IDENTIFIERS	VARCHAR2(500 BYTE)	Yes	(null)	9	(null)
10	OTHER_IDENTIFIERS	VARCHAR2(2500 BYTE)	Yes	(null)	10	(null)

Ilustración 36-Tabla Variation

4. METODOLOGÍA SILE (Search, Identification, Load and Explotation)

Como solución a este desorden de datos genómicos se creó el modelo conceptual para sistemas de información genómicos y éste modelo ha ido evolucionando a lo largo de los años, adaptando e integrando nueva información necesaria para representar la información.

La metodología SILE se basa en un conjunto de actividades que abordar los problemas que proporcionados por los grandes volúmenes de datos, distribuidos en diferentes repositorios y con datos irrelevantes. Vista la problemática actual en el campo de la genómica, es un marco ideal para la aplicación de la metodología SILE.

La metodología SILE se compone de cuatro apartados relacionados entre sí que intentan integrar toda la información relevante de un contexto en un mismo repositorio. La metodología SILE es un acrónimo derivado de las actividades que lo componen: Search Identification, Load y Explotation.

4.1 Search

La primera acción a realizar se basa en la búsqueda de repositorios con información genómica. Como hemos comentado anteriormente la información genómica está distribuida en diversos repositorios que son consultados por genetistas e investigadores. En este proceso de búsqueda de información deberemos analizar y estudiar cuales son los repositorios de información, ver si están actualizados y si son relevantes para nuestro estudio.

Para saber si un repositorio es relevante para nuestro estudio deberemos ver si contiene la información acerca del gen, cromosoma o variación que estemos buscando. Por otra parte deberemos ver si esta información es actualizada, analizaremos si han habido cambios recientes en los datos y las fechas de las últimas actualizaciones. Esta verificación es importante ya que puede que encontremos la información que deseamos en un repositorio, pero, si esta información no se ha actualizado en una serie de años, serán por norma general, datos obsoletos e irrelevantes.

Unos de los principales repositorios de información genómicos con más relevancia y con grandes volúmenes de datos son NCBI, LOVD o UMD.

4.2 Identification

El proceso de identificación se basa en realizar un filtro sobre la información obtenida con el fin de evitar información irrelevante y duplicación en los datos genómicos.

Una vez seleccionados los repositorios de estudio, el siguiente paso se basa en el estudio de los datos genómicos de estos repositorios. De todos los datos de estos los repositorios es necesario que los analicemos con el fin de extraer aquellos datos relevantes o de interés. Por ejemplo, si deseamos realizar la búsqueda de un gen particular, toda aquella información relacionada con otros genes ajenos al nuestro es irrelevante. Por otra parte, los datos genómicos, concretamente las variaciones genómicas, poseen una nomenclatura que las describe, si una variación en cualquiera de los repositorios de información no cumple con este formato, se caracterizará como dato irrelevante para el estudio.

Otro tema a abordar es la repetición de la información. Al haber seleccionado diferentes repositorios de donde extraer la información es lo más probable que gran parte de la información esté repetida. Por ejemplo, si realizamos la búsqueda de las variaciones de un gen conocido como es el BRCA1, las variaciones asociadas a este gen se conocen con precisión, por esto es probable que encontremos estas variaciones duplicadas en diferentes repositorios. Nuestra misión será filtrar la información, seleccionando aquella deseada y previniendo la repetición de datos genómicos con el objetivo de obtener una única instancia de los datos relevantes de estudio.

4.3 Load

Una vez seleccionados los repositorios de información y extraída aquella información relevante el siguiente paso a realizar es el proceso de carga de la información genómica en nuestro repositorio.

Para realizar la carga de información partiremos de nuestro modelado conceptual y dependiendo del tipo de información que deseemos cargar en nuestro repositorio intervendrán unas tablas u otras. Para realizar este proceso de carga disponemos de un framework conocido como genoma.loader. Este framework se encarga de recoger aquella información relevante extraída de los diferentes repositorios y almacenarla en nuestro repositorio de información. El proceso de realización de la carga lo veremos con detalle con el caso de estudio del gen APC en el siguiente capítulo de esta tesis.

Almacenando la información relevante de los diferentes repositorios en un único sistema de información, estamos integrando toda la información genómica en un único repositorio. Esto supone grandes beneficios ya que, se posee información de mejor calidad que en los repositorios por separado, y si se ha de realizar una búsqueda el tiempo invertido es mínimo en comparación a realizar la búsqueda en todos los repositorios.

4.4 Explotation

Por último, una vez almacenados los datos genómicos relevantes en nuestro sistema de información, el siguiente paso es la explotación de estos datos.

Para la explotación de la información se ha realizado la creación del software VarSearch. VarSearch es una aplicación web desarrollada para genetistas e investigadores con el objetivo de ayudarles a manejar la gran cantidad de información genómica. Para ello, VarSearch permite realizar análisis de ficheros .VCF y .SANGER con el objetivo de determinar de una manera eficiente aquellas variaciones encontradas en estos ficheros.

VarSearch es una herramienta implementada en tres capas: La capa de presentación es aquella que interactua el usuario, implementada en HTML5, CSS3 y JavaScript. La capa de servicios provee una independencia de la capa de presentación y ofrece un conjunto de servicios web REST, implementados en Java, con la información de la aplicación en formato JSON. Por último la capa de persistencia esta implementada mediante Hibernate y se encarga de realizar un mapeo entre la capa de servicios y los datos del repositorio genómico.

5. SOLUCIÓN PROPUESTA: INTEGRACIÓN DEL GEN APC AL MODELO CONCEPTUAL

5.1 El gen APC (Adenomatous polyposis coli)

El gen APC (Adenomatous polyposis coli) es un gen supresor tumoral que codifica una proteína que juega un importante papel en la supresión de tumores. La pérdida de su función proteínica puede estar ocasionada por diferentes mutaciones. Ésta pérdida de función facilita la aparición de cáncer colorrectal y provoca la enfermedad denominada poliposis adenomatosa familiar [15]. El gen APC humano se encuentra situado en el brazo largo del cromosoma 5, entre las posiciones 21 y 22, desde la base 112.118.468 hasta la 112.209.532, está compuesto por 21 exones y codifica una proteína formada por 2843 aminoácidos. Ha sido identificado en todos los animales mamíferos a los que se les ha estudiado el genoma [15].

Las mutaciones en la línea germinal del gen APC, es decir las que afectan a las células productoras de gametos, son las responsables de la aparición de poliposis adenomatosa familiar (PAF), enfermedad genética de herencia autosómica dominante, que puede transmitirse a los descendientes y ocasiona entre otras manifestaciones una alta probabilidad de desarrollar cáncer de colon. Por contra, las mutaciones esporádicas, es decir las que afectan a las células somáticas o no germinales, tienen lugar en el 80% de los canceres de colón no hereditario y no son transmisibles a la descendencia

Los Pacientes con PAF se caracterizan por cientos de pólipos adenomatosos colorrectales, con una progresión casi inevitable con el cáncer colorrectal en las décadas tercera y cuarta de la vida. Además de las neoplasias colorrectales, los individuos pueden desarrollar síntomas extracolónicos, entre los que se pueden destacar: pólipos del tracto gastrointestinal superior, hipertrofia congénita del epitelio pigmentario de la retina, tumores desmoides, trastornos de los huesos maxilares en el esqueleto y anomalías dentales.

La mutación más común en el cáncer de colon es la inactivación de APC. Estas mutaciones se pueden heredar, o surgir esporádicamente, con frecuencia como resultado de mutaciones en otros genes que producen inestabilidad cromosómica. Una mutación en APC o β-catenina deben ser seguidas por otras mutaciones para convertirse en cancerosa, sin embargo, en los portadores de una inactivación de las mutaciones APC, el riesgo de cáncer colorrectal a los 40 años es casi del 100%.

Poliposis adenomatosa familiar (FAP) es causada por mutaciones en el gen APC. Actualmente se han identificado más de 800 mutaciones en el gen APC. La mayoría de estas mutaciones causan la producción de una proteína APC que es anormalmente corta y no funcional. Esta proteína no puede suprimir el sobrecrecimiento celular lo cual conduce a la formación de pólipos, que pueden convertirse en cancerosas. La mutación más común en la poliposis adenomatosa familiar es una deleción de cinco bases en el gen APC. Esta mutación cambia la secuencia de aminoácidos en la proteína resultante, empezando en la posición 1309.

Existe otra mutación encontrada en aproximadamente un 6 por ciento de las personas de herencia judía de Ashkenazi (Europa oriental y central). Esta mutación resulta en la sustitución del aminoácido lisina por isoleucina en la posición 1307 (también escrito como I1307K o Ile1307Lys). Este cambio inicialmente estaba considerado inofensivo, pero recientemente se ha demostrado que se asocia con un riesgo de un 10 a un 20 por ciento de cáncer de colon.

Dada la importancia del gen APC y sus consecuencias en la salud de las personas, es un gen de gran interés para genetistas e investigadores. Por esta razón nuestro objetivo principal es integrar la información relevante del gen APC repartida en los diferentes repositorios con el fin de proporcionar información útil y de calidad, reduciendo considerablemente el tiempo empleado en la búsqueda de ésta información a genetistas e investigadores.

5.2 Búsqueda de repositorios genómicos.

Para poder integrar la información relevante del gen APC repartida en los diferentes repositorios debemos seguir los pasos que define la metodología SILE explicada anteriormente.

Acorde a la metodología SILE, la primera acción a llevar a cabo se basa en la búsqueda de repositorios con información genómica. La información genómica está distribuida en diversos repositorios que son consultados por genetistas e investigadores. Debemos identificar que repositorios contienen información relevante y una vez identificados validarlos con especialistas en el campo de la genética.

5.2.1. Búsqueda de repositorios con información del gen APC

Para conseguir integrar la información se ha realizado un amplio estudio sobre los repositorios genómicos. Basándonos en la cantidad de consultas realizadas por los genetistas en los diferentes repositorios, la cantidad de información almacenada en cada uno de estos, la fecha de las últimas actualizaciones y si contenían información del gen APC y sus mutaciones, se han seleccionado los siguientes repositorios relevantes para nuestro estudio:

- NCBI [16]: Es parte de la Biblioteca Nacional de Medicina de Estados Unidos (National Library of Medicine), una rama de los Institutos Nacionales. Está localizado en Bethesda, Maryland y fue fundado el 4 de noviembre de 1988 con la misión de ser una importante fuente de información de biología molecular. Almacena y actualiza la información referente a secuencias genómicas en GenBank, un índice de artículos científicos referentes a biomedicina, biotecnología, bioquímica, genética y genómica en PubMed, una recopilación de enfermedades genéticas humanas en OMIM, además de otros datos biotecnológicos de relevancia en diversas bases de datos.

- GENECARDS [17]: GeneCards es una base de datos del genoma humano que ofrece información genómica, proteómica, transcriptómica, genética y funcional de todos los genes conocidos.

- UMD [18]: Base de datos realizada con el objetivo de proporcionar información actualizada sobre las mutaciones del gen APC. Su objetivo es hacer que la información sea de fácil acceso para cualquier persona interesada en las variaciones genéticas en el gen APC, y para proporcionar estas variaciones y sus hallazgos más recientes.

- LOVD [19]: La base de datos Leiden Variación Open (Lovd) es una herramienta flexible basada en una base de datos de código abierto desarrollada en el Centro Médico Universitario de Leiden en los Países Bajos, diseñado para recopilar y mostrar variantes en la secuencia del ADN. A diferencia de las bases de datos del genoma humano, que muestra información sobre todas las variantes de ADN, LOVDs incluir información acerca de los individuos en los que las variantes se encontraron. Esta base de datos contiene referencias a otras bases de datos consultadas:

    - LOVD: Colon cancer gene variant databases [20]

    - LOVD: Leiden Open Variation Database [21]

    - LOVD: The APC Mutation Database [22]

    - Zhejiang University Center for Genetic and Genomic Medicine [19]

5.2.2. Validación de los repositories genómicos

Una vez realizada la búsqueda de los repositorios que contienen información del gen APC deberemos validarlos con especialistas en el campo de la genómica.

Inicialmente todas estas bases de datos se consideraron relevantes para la integración ya que poseen gran cantidad de información relacionada con el gen y sus mutaciones, además de ser frecuentemente utilizadas y actualizadas.

Con el fin de validad este estudio sobre las bases de datos candidatas para la integración se requirió de genetistas e investigadores. Estos especialistas en el campo genómico revisaron el conjunto de bases de datos preseleccionadas haciendo hincapié en las mutaciones encontradas. Una vez finalizado el estudio muchas bases de datos fueron descartadas ya que no eran lo suficientemente completas en cuanto a la cantidad de variaciones y no estaban recientemente actualizadas.

5.2.3. Selección de repositorios genómicos

Tras finalizar el estudio de búsqueda de los repositorios y la validación de estos por genetistas, las bases de datos a integrar en nuestro sistema de información son: NCBI [16], UMD [18] y LOVD: Colon cancer gene variant databases [20].

5.3 Identificación de información genómica

El segundo paso de la metodolgía SILE es la identificación. Esta identificación implica la identificación de las variaciones de los repositorios seleccionados, así como la transformación de las variaciones, eliminando duplicaciones y redundancias.

5.3.1. Extracción de variaciones

Una vez identificados los repositorios genómicos a integrar en nuestro sistema de información, el siguiente reto a abordar es la identifación de las variacions de los diferentes repositorios, acorde a la metodología SILE. Es necesario identificar las variaciones repetidas y almacenar únicamente una instancia de estas. En la transformación de variaciones, a la hora de generar el fichero en donde unifica todas las variaciones, comprueba que no existan variaciones repetidas.

Algunos repositorios de información ofrecen herramientas y servicios web para poder acceder a su información almacenada. En nuestro caso únicamente NCBI nos proporciona estas herramientas. Sin embargo la mayoría de estos repositorios únicamente ofrecen su información a través de la web, sin la posibilidad de descargar esta información. Este es el caso de los repositorios UMD y LOVD.

Para abordar esta problemática existen técnicas de parseo de webs que permiten obtener el código HTML de la página. En este código HTML encontraremos la información

genómica junto a los lenguajes WEB. Por lo tanto, realizaremos un filtro sobre el código HTML para obtener la información relevante.

Para realizar la carga de información genómica disponemos de un framework denominado Genoma Loader. Este framework esta implementado en Java y está dividido en módulos. Cada uno de estos módulos corresponden a las bases de datos de donde extraemos información, en nuestro caso NCBI y LOVD son bases de datos que tienen su correspondiente módulo, y por lo tanto los reutilizaremos para la carga del gen APC. UMD es una base de datos que no ha sido cargada previamente y requerirá la creación de un nuevo módulo en Genoma Loader

5.3.1.1. Extracción de variaciones de NCBI

La primera base de datos de la cual vamos a extraer las variaciones es NCBI. Como hemos comentario anteriormente NCBI proporciona herramientas y ficheros XML con los que comparte la información a los usuarios.

NCBI almacena las variaciones de los genes en formato XML. Existe un fichero XML para cada cromosoma y dentro de cada fichero XML se almacenan los genes y variaciones correspondientes a ese cromosoma.

Para descargar el XML que contenga el gen APC y sus correspondientes variaciones debemos conocer el cromosoma que ocupa. En este caso el gen APC se encuentra en cromosoma 5. Esta información podemos obtenerla en NCBI.

Conociendo el cromosoma a descargar accedemos al ftp que NCBI ofrece [23]. En este FTP podemos encontrar los ficheros XML organizados por cromosoma. Estos ficheros están comprimidos en formato .xml.gz y pesan entorno a los 2GB comprimidos, por lo que descomprimidos pueden llegar a los 20GB. Este procesamiento no es viable a si no se dispone de buen procesamiento. Por esta razón realizamos la descarga de los ficheros a un servidor dedicado a ello. Esta descarga la realizamos con la siguiente instrucción:

```
wget ftp://ftp.ncbi.nih.gov/snp/organisms/human_9606/XML/ds_ch5.xml.gz
```

Con esta instrucción estamos indicado que deseamos descargar el fichero XML del cromosoma 5 (ds_ch5.xml.gz), en donde encontraremos el gen APC y sus variaciones.

Una vez descargado, debido a su tamaño y a que contiene información irrelevante para nuestro estudio, lanzaremos un script que realizará un filtro sobre este fichero guardando en un nuevo fichero XML únicamente aquellas variaciones que pertenezcan al gen APC.

Este script denominado Verify utiliza un lenguaje XSLT y se encarga de crear un nuevo fichero XML a partir de uno existente seleccionando únicamente el gen deseado. Para ejecutar el script lanzaremos la siguiente instrucción:

java -classpath lib/jdom.jar:lib/saxon9he.jar:bin Verify -o out/ -gi <299116992> -xml ds_ch5.xml.gz

Mediante esta instrucción estamos utilizado las librerías JDOM en nuestro script Verify, teniendo como output un fichero XML (-xml), en el directorio out (-o out/), a partir del archivo ds_ch5.xml.gz, obteniendo únicamente la información del gen que posee el GI 299116992. Este GI corresponde al gen APC y podemos encontrarlo en NCBI.

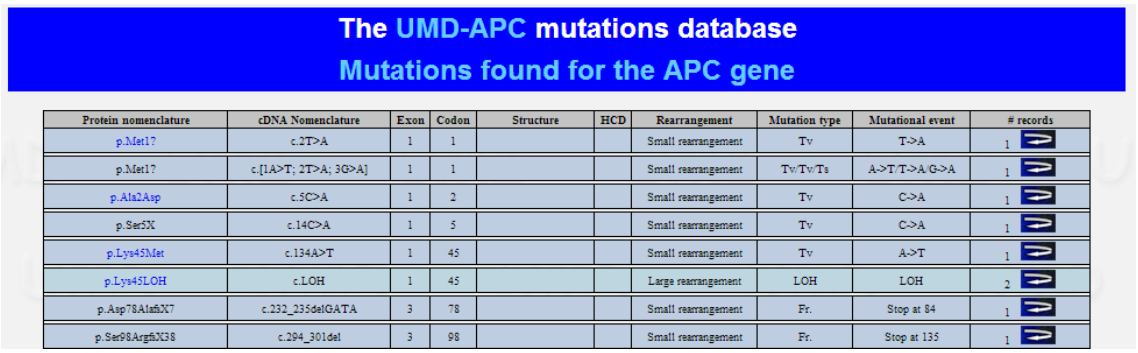
Al haber ejecutado esta instrucción en nuestra terminal obtenemos un fichero XML con el nombre ds_APC.xml que contiene las variaciones del gen APC del repositorio de NCBI. Estas variaciones están en formato XML y posteriormente será transformada mediante el framework Genoma Loader. Esta tarea la realizaremos en el capítulo 6.4

5.3.1.2. Extracción de variaciones de UMD

Una vez extraídas las variaciones de NCBI la siguiente base de datos de la cual extraer las variaciones es UMD.

El repositorio UMD (Universal Mutation Database) que incluye 2043 Mutaciones. Esta información que contiene es compartida a través de los usuarios mediante la web. No proporciona ninguna herramienta (ftp, servicios web, etc.) con la cual podamos acceder a los datos. Con el fin de acceder a las mutaciones, adoptaremos el uso de técnicas de parseo de webs con el fin de obtener los datos de las variaciones en código HTML.

Para obtener el listado de variaciones del gen APC debemos conocer este repositorio, realizando una búsqueda global para obtener todos los resultados. Los resultados obtenidos son representados en una tabla junto a sus referencias bibliográficas [24]. Para cada mutación obtenida en esta tabla ofrecen información acerca de su codificación en proteína, cDNA, Exon, Codon, Structure, HCD, Rearrangement, Mutation type, Mutation event y Records (Referencias).



The UMD-APC mutations database

Mutations found for the APC gene

Protein nomenclature	cDNA Nomenclature	Exon	Codon	Structure	HCD	Rearrangement	Mutation type	Mutational event	# records
p.Met1?	c.2T>A	1	1			Small rearrangement	Tv	T->A	1
p.Met1?	c.[1A>T; 2T>A; 3G>A]	1	1			Small rearrangement	Tv/Tv/Ts	A->T/T->A/G->A	1
p.Ala2Asp	c.5C>A	1	2			Small rearrangement	Tv	C->A	1
p.Ser5X	c.14C>A	1	5			Small rearrangement	Tv	C->A	1
p.Lys45Met	c.134A>T	1	45			Small rearrangement	Tv	A->T	1
p.Lys45LOH	c.LOH	1	45			Large rearrangement	LOH	LOH	2
p.Asp78AlafsX7	c.232_235delGATA	3	78			Small rearrangement	Fr.	Stop at 84	1
p.Ser98ArgfsX38	c.294_301del	3	98			Small rearrangement	Fr.	Stop at 135	1

Ilustración 37- Repositorio UMD

De toda esta información obtenida acerca de una mutación únicamente es relevante

para nuestro estudio el cDNA Nomeclature, que identifica la posición y el cambio realizado en el cromosoma, y sus referencias bibliográficas (Records). Con estos dos valores nos encargaremos de calcular información adicional con el fin de almacenarla en nuestra base de datos.

Para obtener todos estos resultados en código HTML realizaremos clic derecho en la página web y seleccionamos Ver código fuente de la página. Una vez obtenido este código HTML lo almacenaremos en un fichero con el fin de parsear aquella información relevante.

En este código HTML encontraremos la información genómica junto a los lenguajes WEB. Por lo tanto, realizaremos un filtro sobre el código HTML para obtener la información relevante. Identificar la información genómica en el código HTML requiere conocer la sintáxis básica de este lenguaje.

Para ello debemos encontrar en el código el encabezado de la tabla de resultados y a partir de este obtener los resultados. El código del encabezado de la tabla es el siguiente:

```
<tr>
<th bgcolor='#C3C3C3'  class='Style2'> Protein nomenclature </th>
<th bgcolor='#C3C3C3' class='Style2'>cDNA Nomenclature </th>
<th bgcolor='#C3C3C3' class='Style2'>Exon </th>
<th bgcolor='#C3C3C3' class='Style2'>Codon </th>
<th bgcolor='#C3C3C3' class='Style2'> Structure </th>
<th bgcolor='#C3C3C3' class='Style2'> HCD </th>
<th bgcolor='#C3C3C3' class='Style2'> Rearrangement </th>
<th bgcolor='#C3C3C3' class='Style2'> Mutation type </th>
<th bgcolor='#C3C3C3' class='Style2'> Mutational event </th>
<th bgcolor='#C3C3C3' width='100' class='Style2'> # records </th>
</tr>
```

Partiendo de la estructura que presenta el HTML podemos identificar que aquellas etiquetas HTML entre las cuales encontraremos la información. Identificando que es una tarea repetitiva, se ha implementado la clase ***ConvertFile*** encargada de generar un nuevo fichero que contenga únicamente los campos relevantes para nuestro estudio, que en este caso son el cDNA Nomeclature y los Records.

ConvertFile se encarga de leer cada línea del fichero que contiene el código HTML. Si la línea del fichero extraída contiene “<th bgcolor='#C3C3C3' class='Style1'>”, que hemos identificado en los resultados de la tabla, procederemos a extraer su campo cDNA mediante el método TakecDNA. En caso de que la línea del fichero extraída contenga "<td class='Style1'> <CENTER><A HREF estaremos obteniendo la referencia bibliográfica. Para cada cDNA o referencia encontrada la escribimos en un fichero cdna+href.txt que contendrá la información de las variaciones del gen APC sin código HTML. De esta forma posteriormente podremos leer esta información de manera eficiente para su transformación en inserción el repositorio.

5.3.1.3. Extracción de variaciones de LOVD

El repositorio genómico LOVD, al igual que UMD, es un repositorio utilizado para la consulta, pero no para la extracción de información. No proporcionan un soporte para la extracción de información y por lo tanto deberemos obtener las variaciones mediante la extracción del código HTML.

Los resultados obtenidos en este repositorio [25] están muy definidos ya que proporcionan mucha información acerca de las variaciones encontradas. Para nuestro estudio no es necesária tanta información ya que a partir del DNA podemos calcular toda la información.

Extraer información de bases de datos de LOVD es una tarea previamente realizada e implementada en el framework Genoma Loader. Existe un módulo de extracción de variaciones que recibe como input la URL del repositorio y la información que se desea extraer y genera como output un fichero XML con la información de las variaciones indicadas.

Este módulo posee un fichero lovd.properties encargado de recoger la URL del repositorio y la información que se desea extraer. En nuestro caso nuestro repositorio es LOVD: Colon cancer gene variant databases [20]:

```
APC_URL=http://chromium.liacs.nl/LOVD2/colon_cancer/variants.php?action=searc
h_unique&limit=1000&order=Variant%2FDNA%2CASC&page=1
```

Mediante esta variable indicamos la página web en donde hemos encontrado el resultado de la búsqueda de variaciones del gen APC en la base de datos LOVD. Con el objetivo de indicar la información a extraer almacenamos la siguiente variable:

```
APC_fields_order=Exon,DNA_change,,,,,,,,DB-ID,,,,
```

En la tabla de las variaciones de LOVD [26] encontramos 16 columnas con información adicional de cada variación encontrada. Para indicar las columnas relevantes para nuestro estudio escribimos su nombre en la variable APC_fields_order, y para las que no nos interesan las identificamos con el carácter ",". De esta forma estamos indicado la extracción de información de la columna Exon, Dna_change y DB-ID (referencias bibliográficas).

Una vez indicado el repositorio y los campos de interés, el siguiente paso es extraer las variaciones. Con el fin de extraer las variaciones se ha implementado la clase ***Lovd***.

Esta clase se encarga de obtener las variaciones del gen APC en el repositorio seleccionado a través de una petición HTTP/GET. De esta forma obtiene la información de la página web en una variable, eliminado el código HTML y extrayendo los campos Exon, DNA_change y DB-ID. Toda esta información obtenida es exportada en un fichero XML. Este XML contendrá la información de las variaciones del gen APC de LOVD. De esta forma

posteriormente podremos leer esta información de manera eficiente para su transformación en inserción el repositorio.

5.3.2. Transformación de variaciones

En el capítulo anterior hemos extraído de los diferentes repositorios las variaciones del gen APC. Esta información está distribuida en tres ficheros diferentes, cada uno correspondiente a la base de datos de la cual está extraída. La información que poseen los diferentes ficheros generados es información no estructurada. A lo largo de este capítulo se veremos como estructurar la información mediante la creación de objetos, a partir de la información de cada uno de los ficheros, y unificarlos todos en un mismo fichero APC.dmp para su posterior inserción en la base de datos.

La transformación de variaciones en un proceso mediante el cual información no estructurada, extraída de una base de datos, se estructura mediante objetos, calculando información adicional, y dejándola preparada para su inserción.

Para realizar la transformación de variaciones el framework Genoma Loader provee de un módulo que analiza los ficheros, creando objetos y añadiéndolos a un fichero con formato .dmp

Otro objetivo que aborda la transformación de variaciones es la eliminación de variaciones duplicadas. Acorde al proceso Identification de la metodología SILE, muchas variaciones pueden estar duplicadas en diferentes repositorios. Es necesario identificar las variaciones repetidas y almacenar únicamente una instancia de estas. En la transformación de variaciones, a la hora de generar el fichero en donde unifica todas las variaciones, comprueba que no existan variaciones repetidas.

Para la transformación existen diferentes módulos funcionales en Genoma Loader. Cada módulo es específico para un repositorio genómico. En cada módulo se implementan las clases Creator y Mutation. Estas clases proporcionan la creación de objetos a partir de los ficheros anteriores. Cada módulo se encarga de leer el fichero de las variaciones extraídas, recorriendo cada registro obteniendo el DNA y las referencias bibliográficas. A partir del DNA y las referencias bibliográficas se calcula la información de la vista de variaciones, vista de fuente de datos y la vista de bibliografía de nuestro esquema relacional.

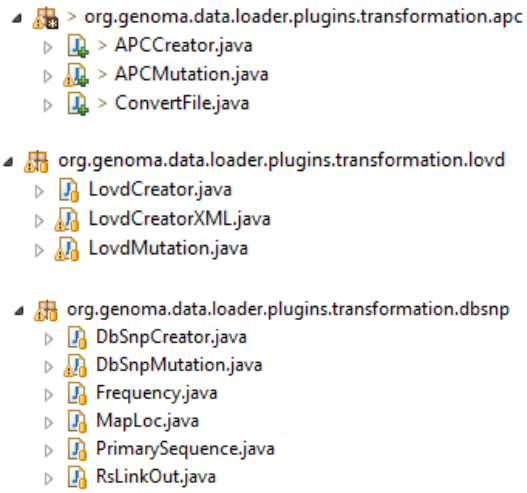


Ilustración 38-Genoma Loader

Con el fin de transformar las diferentes variaciones extraidas Genoma Loader provee estos módulos que son ejecutados mediante argumentos en nuestro entorno de desarrollo. El comando utilizado para pasar los parámetros es el siguiente:

--tool <Action> --plugin <pluginName> --gene <Gene>

Mediante esta instrucción ejecutamos la acción deseada, utilizando el plugin necesario y el gen correspondiente. Esta instrucción genérica nos permite ejecutar la transformación de la base deseada teniendo en cuenta el gen. Las instrucciones utilizadas para cargar los tres repositorios son los siguientes:

--tool transformation --plugin dbSNP --gene APC

--tool transformation --plugin lovd --gene APC

--tool transformation --plugin apc --gene APC

Cuando se ejecuta alguna de estas instrucciones, independientemente del módulo a ejecutar, la clase que se instancia es ***TSVMutationTransformer***.

Esta clase se encarga de recoger el fichero de variaciones a transformar a partir del parámetro de entrada. Una vez obtenido el fichero de variaciones se encarga de leer cada uno de los registros y crear objetos con esta información. El método encargado de realizar este proceso es el método evaluate de la clase AtsvCreator.

ATsvCreator se encarga de instanciar un objeto de la clase ***ARawMutation*** que toma como argumento la línea del fichero leído. ***ARawMutation*** calcula toda la información acerca del registro del fichero obtenido.

ARawMutation se encarga de interpretar la variación obtenida del fichero. A partir del DNA calcula el tipo de especialización, su localización génica y cromosómica, fenotipos,

alineamiento, su flanking de nucleótidos entre los que se encuentra la variación, la secuencia insertada y la deleción de bases de nucleótidos. A parte de toda esta información, también almacena las referencias bibliográficas obtenidas en la extracción de variaciones.

Una vez transformados todos los registros del fichero el siguiente paso es almacenarlos en un mismo archivo con extensión .dmp. Esta transformación de variaciones se realiza sucesivamente para todos los ficheros extraídos de las diferentes bases de datos y se integran en este mismo fichero. A la hora de almacenar las variaciones en este nuevo fichero generado se comprueban que no exista ninguna variación duplicada. En caso de que se encuentre una duplicación se eliminan dejando únicamente una instancia de esta. La clase ***AmutationTransformer*** proporciona la generación del fichero APC.dmp con las variaciones transformadas.

AmutationTransformer obtiene para cada fichero sus variaciones transformadas. Comprueba que de estas variaciones transformadas no existan duplicaciones, y en caso de que existan, eliminarlas mediante el método removeDuplicates. Eliminadas las duplicaciones el siguiente paso es serializar estas variaciones en el nuevo fichero APC.dmp, realizado en el método serialize.

Tras finalizar la transformación de variaciones de los diferentes repositorios hemos obtenido un fichero APC.dmp que posee toda la información de las variaciones de estos repositorios junto a sus referencias bibliográficas.

Una vez obtenido este fichero el siguiente paso se basa en la carga de la parte estructural y la carga de estas variaciones.

5.4 Carga de información genómica

El tercer proceso de la metodología SILE es la carga (Load). La carga de información genómica, extraida y transformada en los procesos anteriores se divide en la carga de la parte estructural y la carga de variaciones. En este proceso de carga se implementa la extensión de múltiples referéncias bibliográficas, acorde a la propuesta de esta tesis.

5.3.3. Carga de la parte estructural

Una vez abordados los procesos de Search e Identification de la metodología SILE la siguiente tarea a realizar es la carga de la información extraída. Este proceso Load de la

metodología SILE parte de las variaciones extraídas previamente y procede a su inserción en la base de datos.

En nuestro caso de estudio las variaciones extraídas están asociadas al gen APC. Inicialmente en nuestro sistema de información no disponemos de información del gen APC sobre la cual asociar estas variaciones. Por lo tanto es necesaria la carga de la información de la parte estructural del gen APC.

La parte estructural posee información relacionada con el gen, su símbolo, alelos transcritos y exones. La carga CORE o parte estructural hace referencia a la carga de las tablas de nuestro modelo conceptual:

- Gene
- Allele
- Allele_Databank_Ident
- AllelicReferenceType
- Transcript
- Exon
- Exon_Transcript

La parte estructural de cada uno de los genes es descargada de NCBI. Esta base de datos proporciona unas herramientas que permiten la descarga de sus datos mediante un conjunto de librerías y su correspondiente API.

Para ejecutar esta funcionalidad del framework Genoma Loader debemos indicar el gen que deseamos cargar y su correspondiente NG (versión del gen). Con el objetivo de poder realizar esta acción simultánea para diversos genes, hemos de crear un fichero ngGenes.txt que contenga todos los genes que deseemos cargar y su correspondiente NG. En nuestro caso únicamente vamos a cargar el gen APC, por lo que el fichero contendrá la siguiente información:

APC NG_008481

Partiendo de este fichero, nuestro framework se encarga de recuperar su contenido obteniendo, para cada gen, la información de la base de datos de NCBI, obtenida a través de las librerías que proporciona.

Con el objetivo de cargar esta información se ha implementado la clase ***LoadCore***, basándose en la API de NCBI. ***LoadCore*** es el método encargado de utilizar las librerías EUtilsServiceStub de NCBI. Estas librerias nos proporcionan toda la información estructural del gen deseado. ***LoadCore*** se encarga de recuperar el contenido del fichero ngGene.txt obteniendo para cada gen y su correspondiente NG la información necesaria para cargar la parte estructural. Estas librerias se basan en llamadas a servicios web de NCBI y nos proporcionan de cada gen su Allele, Allele_Databank_Ident, AllelicReferenceType, Transcript, Exon y Exon_Transcript. Una vez obtenida esta información y relacionada con nuestro gen, realizamos la inserción de la información en nuestro repositorio de información.

La carga de la parte estructural es común a los diferentes repositorios, únicamente hemos de ser conscientes la versión con la que estamos trabajando. En nuestro caso la versión del NG con la que trabajamos es común a todas las versiones extraidas. Si las versiones de NG difiriesen habría que realizar un alineamiento de secuencias [27] con el objetivo de hacer coincidir los cambios de las mutaciones en las diferentes versiones.

Una vez extraída y almacenada la información estructural del gen APC es el momento de insertar las variaciones extraídas y asociarlas a esta información del gen cargada.

5.3.4. Extensión e implementación del proceso de carga

La aportación que se realiza en esta tesis es la extensión e implementación del proceso de carga. El modelado conceptual previamente explicado da soporte al almacenamiento de múltiples referencias bibliográficas para una misma variación. A pesar de estar modelado, en la implementación del framework Genoma Loader no estaba soportado. Las bases de datos previamente cargadas únicamente contenían una referencia bibliográfica por variación y no se había dado el caso de múltiples referencias. Este fallo de implementación lo encontramos a la hora de cargar las variaciones de la base de datos UMD. UMD presenta en muchos casos entre 3 y 10 referencias bibliográficas por variación. Al no haberse dado este caso previamente no se ha podido detectar, y en el proceso de inserción de variaciones detectamos esta necesidad.

Para dar soporte a las múltiples referencias es necesario modificar el modelo de implementación de Genoma Loader, modificando las clases ***Variation*** y ***Precise***, añadiéndoles un atributo que contiene una lista de bibliografías de referencia y su constructor asociado.

Una vez modificados las clases del modelo de Genoma Loader, el siguiente paso es modificar el metodo de inserción de variaciones en la base de datos para insertar las múltiples referencias. Este método de inserción será utilizado en el siguiente capítulo de inserción de variaciones. El Metodo MultipleVariationRef recibe la información de cada una de las variaciones extraidas y sus multiples referencias. En el caso que encuentre una variacion que posea diveras referencias bibligráficas, en vez de insertar una única vez se insertará tantas veces como referencias posea la variación.

Una vez adaptado el framework para dar soporte a las múltiples referencias de una variación, el siguiente paso es la inserción de las variaciones en nuestra base de datos.

5.3.5. Carga de variaciones.

Una vez extraídas las variaciones y almacenada la parte estructural del gen APC, con el fin de abordar el proceso Load de la metodología SILE, el siguiente paso es la inserción de las variaciones del gen APC extraídas de los diferentes repositorios.

Como hemos explicado previamente, las variaciones extraídas y transformadas se encuentran en el fichero APC.dmp. Partiendo de este fichero de variaciones transformadas nuestro framework Genoma Loader se encargará de insertar esta información y asociarla al gen APC. Con el fin de abordar esta funcionalidad se ha implementado la clase ***MutationLoader***

MutationLoader es la clase utilizada como punto de partida para la inserción del fichero de variaciones extraido. Su función es encontrar el fichero de variaciones APC.dmp en nuestro directorio, cargar las variaciones y las referencias en memoria, procesarlas e insertarlas. El proceso e inserción de variaciones se realizan en una clase ***SqlOutManager*** encargada de calcular información adicional para cada una de las variaciones del fichero, obteniendo su Fenotipos, especializaciones en Precise e Imprecise y su asociación con el Gen APC previamente insertado.

Los parámetros de entrada utilizados para la inserción de variaciones son subministrados mediante argumentos en nuestro entorno de desarrollo. El siguiente comando permite la ejecución de esta funcionalidad:

--tool load --plugin dbSNP --plugout Oracle

Mediante este commando estamos indicado que deseamos realizer un proceso de carga (load) en el módulo de carga (plugin dbSNP) y con almacenamiento en nuestro repositorio de información (plugout Oracle)

Una vez realizado el proceso de inserción toda la información del gen APC, sus variaciones, sus referencias bibliográficas y sus relaciones, están almacenados en nuestro repositorio genómico. Como podemos comprobar, en uno de los registros de la tabla Gene de nuestra base de datos está almacenado correctamente el gen APC (Homo sapiens adenomatous polyposis coli).

CHR_ELEM_ID	ID_SYMBOL	ID_HUGO	OFFICIAL_NAME
531	APC	583	Homo sapiens adenomatous polyposis coli

Ilustración 39-Gen APC almacenado

5.5 Explotación de información genómica

Una vez almanceda la información genómica extraída de los diferentes repositorios, el siguiente paso es la explotación de esta información a los usuarios.

Para la correcta explotación de información se ha desarrollado el software VarSearch. VarSearch es una aplicación web desarrollada para genetistas e investigadores con el objetivo de ayudarles a manejar la gran cantidad de información genómica.

Estos genetistas e investadores en su proceso de análisis de ADN, secuencian una muestra de un paciente obteniendo como resultado un fichero con extensión .VCF o .SANGER. Estos ficheros contienen las variaciones identificadas en el paciente. A partir de estos ficheros, los genetistas e investigadores, buscan de forma manual cada una de las variaciones del fichero en los repositorios genómicos. El coste de este proceso se ve afectado por el número de variaciones encontradas en el fichero.

Con el objetivo de facilitar esta tarea a genetistas e investigadores, VarSearch permite realizar análisis de ficheros .VCF y .SANGER con el objetivo de determinar de una manera eficiente aquellas variaciones encontradas en estos ficheros. Para ello se respalda en el Sistema de información genómico mencionado anteriomente, obteniendo cada una de las variaciones del fichero y realizando la búsqueda en nuestro repositorio de información genómica. De esta manera ofrece a los usuarios aquellas variaciones encontradas con un coste minimo.

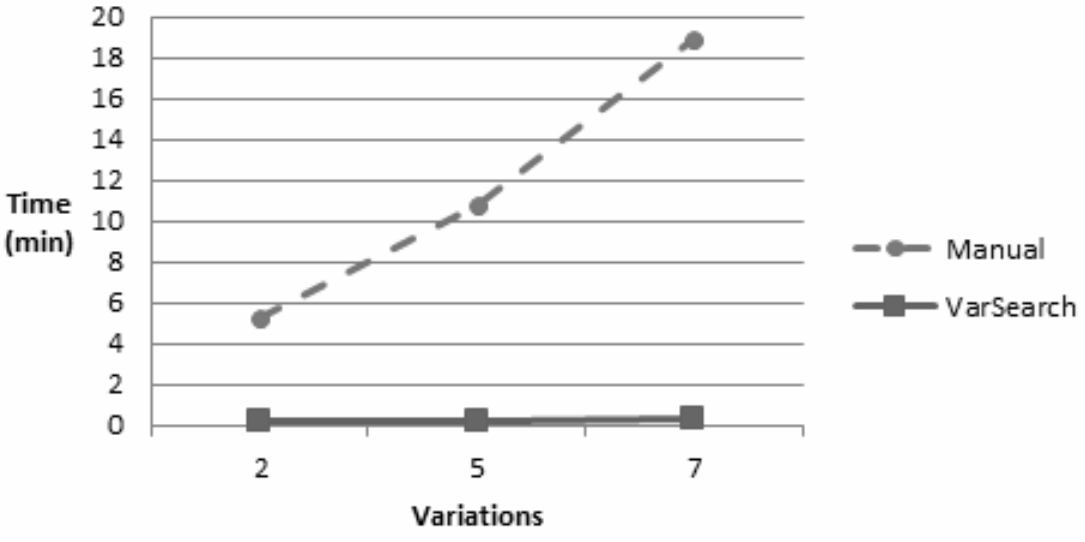


Ilustración 40-VarSearch Coste

Como podemos observar en la Figura 40, El coste de realizar una búsqueda manual se asciende a 5'32 minutos por la búsqueda de 2 variaciones, 10'83 minutos para 5 variaciones y 18'89 minutos para 7 variaciones. Sin embargo, para la búsqueda realizada por VarSearch permanece constante entre 2 y 3 segundos para diferentes variaciones.

6. CONCLUSIONES Y TRABAJO FUTURO

En esta tesis se ha presentado el diseño y desarrollo de un sistema de información genómico que integra datos heterogéneos, procedentes de fuentes externas, almacenando la información relevante. En el proceso, se ha demostrado las ventajas que proporciona una aproximación metodológica basada en técnicas de modelado conceptual, al poder abordar los problemas de heterogeneidad, dispersión y desestructuración de dichas fuentes.

Durante el desarrollo de esta tesis hemos inferido en los procesos realizados, para las diferentes actividades, pudiendo extraer los siguientes conceptos y conclusiones:

- La información genómica no esta bien estructurada. Existen multiples repositorios dispersos, con datos heterogéneos difícilmente de identificar y de unificar.

- La incorporación de las múltiples referéncias bibliográficas para una variación aporta un gran valor a la información. De esta forma se garantiza que una variación es validada por diferentes repositorios adquiriendo gran relevancia.

- Identificar los diferentes repositorios de donde extraer la información es una tarea compleja. Requiere conocer las fuentes de datos, como buscar su contenido, identificar si están actualizadas y si su contenido es válido. Mediante la metodología SILE se posee un procedimiento estructurado a la hora de abarcar la información de los múltiples repositorios.

- La integración de la información en un único modelo conceptual es esencial para conocer información valida y de calidad.

Las aportaciones concretas de este proyecto han sido:

- Se ha completado el diseño de un modelo conceptual que representa y unifica el conocimiento que los científicos tienen actualmente sobre el genoma. El modelado se ha abordado siguiendo una técnica de vistas que representan parcelas diferenciadas del dominio: estructura del genoma, transcripción, variaciones, rutas metabólicas y fuentes de datos y referencias bibliográficas.

- Se ha realizado la carga de un nuevo gen de importancia científica, el gen APC, junto con la carga del nuevo repositorio genómico UMD. Este repositorio no había sido cargado previamente y en el detectamos multiples referencias bibliográficas.

- Se ha realizado una extensión del modulo de carga dando soporte a las múltiples referencias que puede presentar una variación. En nuestro caso de estudio hemos detectado esta deficiencia y la hemos implementado de forma modular para futuros repositorios con múltiples referéncias bibliográficas sobre una variación.

A lo largo del proceso de desarrollo de esta tesis se han realizado los objetivos previstos mediante el uso de diferentes aplicaciones, entornos de desarrollo, frameworks y procesos. Gracias a estos procesos de modelado, desarrollo, carga e implementación se han podido detectar posibles mejoras a realizar con el objetivo de alcanzar estos procesos de una forma más eficiente.

Como trabajos futuros se contemplan:

- La extensión del modelo para nuevas áreas del dominio, por ejemplo incorporando la representación de la asociación fenotipo-genotipo, incorporando información relativa a genomas reales de individuos particulares e incorporando información relativa a tratamientos conocidos.

- La posibilidad de implementar la base de datos actual con tecnologías No SQL, ya que éstas están concebidas para soportar un volumen de datos mucho mayor que en el caso de las bases de datos relacionales.

- Optimización del proceso de extracción de las variaciones de los diferentes repositorios, mediante una búsqueda y un parseo de web automático apartir de un gen.

- Optimización del proceso de carga de variaciones, mejorando el rendimiento con procesos almacenados y optimización de las consultas de inserción y recuperación de información.

- Explotación de la información almacenada en nuestro repositorio genómico.

- Almacenamiento de nuestro sistema de información genómico en la NUBE.

Estas mejoras propuestas se consideran como trabajo futuro ya que no son los objetivos de esta tesis. Estos trabajos se contemplan con el fin de mejorar el proceso global que abarca esta tesis y con el objetivo de reducir el tiempo de asociado a la carga de repositorios genómicos y todo lo que ello conlleva.

7. REFERENCIAS

[1] Oscar Pastor, Ana M. Levin, Juan Carlos Casamayor, Matilde Celma, Aremy Virrueta, Luis E. Eraso and Manuel Perez-Alonso, "Enforcing Conceptual Modeling to Improve the Understanding of Human Genome",2011.

[2] E. Bornberg-Bauer and N. W. Paton, "Conceptual data modelling for bioinformatics," *Briefings in bioinformatics,* vol. 3, p. 166, 2002.

[3] K. Garwood, C. Garwood, C. Hedeler, T. Griffiths, N. Swainston, S. G. Oliver, and N. W. Paton, "Model-driven user interfaces for bioinformatics data resources: regenerating the wheel as an alternative to reinventing it," *BMC bioinformatics,* vol. 7, p. 532, 2006.

[4] N. W. Paton, S. A. Khan, A. Hayes, F. Moussouni, A. Brass, K. Eilbeck, C. A. Goble, S. J. Hubbard, and S. G. Oliver, "Conceptual modelling of genomic information," *Bioinformatics,* vol. 16, p. 548, 2000.

[5] Orlando Valega, Don Guillermo, “Gregor Mendel’s Laws of heredity - Mendelian theory”

[6] M. Harris, J. Clark, A. Ireland, J. Lomax, M. Ashburner, R. Foulger, K. Eilbeck, S. Lewis, B. Marshall, and C. Mungall, "The Gene Ontology (GO) database and informatics resource," *Nucleic acids research,* vol. 32, p. D258, 2004.

[7] E. S. Lander, L. M. Linton, B. Birren, C. Nusbaum, M. C. Zody, J. Baldwin, K. Devon, K. Dewar, M. Doyle, and W. FitzHugh, "Initial sequencing and analysis of the human genome," *Nature,* vol. 409, pp. 860-921, 2001.

[8] Kornberg A, “Dna replication”, vol. 263, no1, pp. 1-4, 1988.

[9] O. Pastor, "Conceptual Modeling Meets the Human Genome," *Conceptual Modeling-ER 2008,* pp. 1-11, 2008.

[10] O. Pastor, van der Kroon,M., Levin,A., Casamayor,J.C., Celma,M., "A Conceptual Modeling Approach to Improve Human Genome Understanding. ," *Handbook of Conceptual Modeling: Theory, Practice and Research Challenges. In Embley,D., Thalheim,B. (Eds.), Springer,* pp. 517-541, 2011.

[11] O. Pastor, A. Levin, M. Celma, J. Casamayor, A. Virrueta, and L. Eraso, "Model-Based Engineering Applied to the Interpretation of the Human Genome," *The Evolution of Conceptual Modeling,* pp. 306-330.

[12] K. Paigen and P. Petkov, "Mammalian recombination hot spots: properties, control and evolution," *Nature Reviews Genetics,* vol. 11, pp. 221-233.

[13] Stefan Schuster, David A. Fell & Thomas Dandekar, “A general definition of metabolic pathways useful for systematic organization and analysis of complex metabolic networks”, Nature Biotechnology 18, 326 - 332 (2000)

[14] Peter Pin-Shan Chen, “The entity-relationship model—toward a unified view of data”, Volume 1 Issue 1, 1976

[15] Paul Polakis, “The adenomatous polyposis coli (APC) tumor suppressor“, Volume 1332, Issue 3, 7 June 1997

[16] http://www.ncbi.nlm.nih.gov/gene/324 , 3 de Abril de 2013

[17] http://www.genecards.org/cgi-bin/carddisp.pl?gene=APC#snp , 3 de Abril de 2013.

[18] http://www.umd.be/APC/ , 7 de Abril de 2013.

[19] http://www.lovd.nl/3.0/home , 7 de Abril de 2013.

[20] http://chromium.liacs.nl/LOVD2/colon_cancer/home.php?select_db=APC , 9 de Abril de 2013.

[21] http://databases.lovd.nl/genomed/genes/APC , 9 de Abril de 2013.

[22] http://genomed.org/LOVD/HNPCC/home.php?select_db=APC , 9 de Abril de 2013.

[23] ftp://ftp.ncbi.nlm.nih.gov/snp/organisms/human_9606/XML/ , 11 de Abril de 2013.

[24] http://www.umd.be/APC/4DACTION/WS_SC1 , 13 de Abril de 2013.

[25] http://chromium.liacs.nl/LOVD2/colon_cancer/variants.php?action=search_unique&select_db=APC , 13 de Abril de 2013.

[26] http://chromium.liacs.nl/LOVD2/colon_cancer/variants.php?action=search_unique&limit=1000&order=Variant%2FDNA%2CASC&page=1, 13 de Abril de 2013

[27] http://blast.ncbi.nlm.nih.gov/Blast.cgi , 22 de Abril de 2013

APÉNDICE

ConvertFile.java

```
public class ConvertFile {

	boolean onlyFirstTag = true;
	boolean entramos = false;
	String lastEntry;
	int primeravez=0;
	public String getlastEntry() {return lastEntry;}
	public void setlastEntry(String value) {lastEntry = value;}
	public boolean getEntramos() {return entramos;}
	public void setEntramos(boolean value) {entramos = value;}
	public boolean getonlyFirstTag() {return onlyFirstTag;}
	public void setonlyFirstTag(boolean value) {onlyFirstTag = value;}


	public ConvertFile(){};

	public void convert(File f) throws IOException
	{
		File archivo = null;
	    TextFileReader tfr = null;

	  FileWriter fichero = new
FileWriter("/Users/Jorge/data/raw/APCDB/cdna+href.txt");

	  archivo = f;
	  tfr = new TextFileReader (archivo);

	  String line = tfr.readLine();


	  String hrefid;
	  Boolean repeatEntry = false;
	  String lastEntry="";

	      while (line != null) {

			if (line.contains("<td bgcolor=")
					&& line.contains("class='Style1'><center>")
					&& getonlyFirstTag()) {
				takecDNA(getEntramos(), line, fichero);
			}

			if (line.contains("<td class='Style1'><CENTER><A HREF="))
			{

				if (lastEntry.equals(""))
					repeatEntry = true;
				else if (line.contains(lastEntry)) {
					repeatEntry = false;
					setonlyFirstTag(true);
				} else
					repeatEntry = true;
```

```
                        if (repeatEntry) {
                              int ini = line.indexOf("HREF='..");
                              int fin = line.indexOf("' TARGET");
                              hrefid = line.substring(ini + 8, fin);
                              lastEntry = line.substring(0, fin);
                              fichero.write("http://www.umd.be/APC" +
                                          hrefid);
                              System.out.println("Leido href: " + hrefid);
                        }
                  }

                  line = tfr.readLine();
            }
            fichero.close();

      }
public void takecDNA(Boolean entramos, String line, FileWriter fichero)
throws IOException{
             String cDNA;

            if (entramos) {
                  int ini = line.indexOf("<center>");
                  int fin = line.indexOf("</center>");
                  cDNA = line.substring(ini + 8, fin);
                  String cDNA2 = cDNA.replaceAll("&gt;", ">");
                  setonlyFirstTag(false);
                  if (primeravez == 0) {
                        fichero.write(cDNA2 + "\t");
                        primeravez = 1;
                  } else
                        fichero.write("\n" + cDNA2 + "\t");
                  setEntramos(false);
            } else {
                  setEntramos(true);
            }
      }


}
```

Lovd.java

```
public class Lovd {

      static Properties p = new Properties();
      static String gene = "";
      static String[] xmlFieldsOrder;
      static {
            try {
                  p.load(Lovd.class.getResourceAsStream("lovd.properties"));
                  xmlFieldsOrder =
                  ((String)p.get("XML_fields_order")).split(",");
            } catch (IOException e) {
                  System.err.println("ERROR en carga de lovd.properties");
```

```
                e.printStackTrace();
            }
        }
        static String procesaVariacion(String varStr,String[] etiq_campos) {
            String content="";
            content += "\t<Variation>\n";
                LinkedHashMap<String, String> fields = new
        LinkedHashMap<String,  String>(xmlFieldsOrder.length);

            String[] campos = varStr.split(p.getProperty("campo_regexp"));
            for (int i=0; i<etiq_campos.length; i++) {
                if (! "".equals(etiq_campos[i])) {
                    int _arroba = etiq_campos[i].indexOf("@");
                    if (_arroba>0) {
                        String etiq[] =  etiq_campos[i].split("@");
                        int idx = campos[i+1].indexOf(etiq[1]);
                        if (idx>0) {
                            if(etiq[0].equals("Variant
                                                reference"))
                                etiq[0]="Reference";

        if(campos[i+1].substring(idx).split("\"")[1].contains("rs=rs"))
                            {

                                fields.put(etiq[0]+"_"+etiq[1],
(campos[i+1].substring(idx).split("\"")[1]).replace("&", "&amp;"));
                            }

                            else fields.put(etiq[0]+"_"+etiq[1],
campos[i+1].substring(idx).split("\"")[1]);

                        }
                        String[] c = campos[i+1].split("<.*?>");
                        int c_idx=(-1);
                        while (c[++c_idx].matches("[ \t]*"));
                        fields.put(etiq[0],c[c_idx]);
                    } else

        fields.put(etiq_campos[i],campos[i+1].split("<")[0]);
                }
            }
            for (String f : xmlFieldsOrder)
                content += "\t\t<"+f+">"+fields.get(f)+"</"+f+">\n";
            content +="\t</Variation>\n";
            return content;
        }
        /**
         * @param args
         * @throws Exception
         */
        static String getHtmlContent(URL url) throws Exception {
            HttpURLConnection connection =
            (HttpURLConnection)url.openConnection();
            connection.setRequestMethod("GET");
            connection.connect();
            InputStream is = (InputStream)connection.getContent();
            return new String(IOUtils.readFully(is, -1, true));

        }
```

```
	static String getHtmlContent(FileInputStream file) throws Exception {
		byte[] b = new byte[file.available ()];
		file.read(b);
		file.close ();
		return new String (b);
	}

	static String getHtmlContent(String source) throws Exception {
		if (source.contains("http"))
			return getHtmlContent(new URL(source));
		else if (source.contains(".htm"))
			return getHtmlContent(new FileInputStream (source));
		else {
			String gene="";
			try {
				gene = p.getProperty(source+"_URL");
			} catch (Exception e) {
				System.err.println("ERROR: Not found URL for gene
				"+source);
				throw e;
			}
			return getHtmlContent(new URL(gene));
		}
	}
	public static String getLovdXml(String html) {
		String xml="";
		String[] resto = html.split(p.getProperty("gene_location"));
		gene = resto[1].substring(0,resto[1].indexOf("\""));
		xml += ("<LOVD gene=\""+gene+"\">\n");
		resto = resto[1].split("entries per page");
		resto = resto[1].split(p.getProperty("variation_ini"));

		for (int i=1; i<resto.length-1; i++) {
			xml +=
procesaVariacion(resto[i],p.getProperty(gene+"_fields_order").split(","));
		}
		xml += "</LOVD>";
		return xml;
	}
	public static void main(String[] args) throws Exception {
		if (args == null || args.length < 1 || args.length > 2) {
			System.out.println("USAGE: Lovd
<input_filename|gene|lovd_URL> [output_dir]");
			return;
		}
		String html=getHtmlContent(args[0]);
		String xml = getLovdXml(html);
		if (args.length==2) {
			String fileName = args[1]+"/"+gene+"_LOVD.xml";
			FileWriter fw = new FileWriter(fileName);
			fw.append(xml);
			fw.close();
			System.out.println("Output written in "+fileName);
		} else
			System.out.println(xml);
	}

}
```

TSVMutationTransformer.java

```
public class TSVMutationTransformer {

	/**
	 * in case of TSV files, the content usually doesn't start the first
line.
	 * Rather the first line is reserved for headers, and thus the content
will
	 * actually start on line 2. This attribute allows for specifying
where
	 * exactly the content starts.
	 */
	private int contentStart;

	@Override
	public boolean initialize(Map<String, String> params) {
		this.parameters = new HashMap<String, String>(params);
		plugindir = params.get("PLUGINDIR");
		objectCreator = params.get("creator");
		contentStart = Integer.parseInt(params.get("contentstart"));
		return true;
	}

	/**
	 * this method uses the CSVReader library to read a tsv file, as is
output
	 * by the extraction layer of the genoma.data.loader program. A custom
line
	 * number from which to start reading can be specified by passing the
	 * contentStart parameter.
	 *
	 * @param file
	 *            the path to the sourcefile
	 * @param contentStart
	 *            which line number content starts
	 *
	 * @return a List<A> containing the distinct variations as objects
	 * @throws IOException
	 *             Signals that an I/O exception of some sort has
occurred. This
	 *             class is the general class of exceptions produced by
failed
	 *             or interrupted I/O operations.
	 */
	@Override
	public List<ARawMutation> loadFile(AObjectCreator objectCreator,
String geneId) throws IOException {
		File file = new File(getRawDataDir(), geneId + ".muts");
		List<ARawMutation> variations = new ArrayList<ARawMutation>();
		int counterSuccessful = 0;
		objectCreator.initialize(parameters);
		CSVReader reader;


	if(geneId.equals("MSH2")||(geneId.equals("MSH6"))||(geneId.equals("MLH
1")))
		{
```

```
                    reader = new CSVReader(new FileReader(file), ',', '\0',
contentStart);
             }
             else reader = new CSVReader(new FileReader(file), '\t', '\0',
contentStart);

             String[] nextLine;
             String next;

             while ((nextLine = reader.readNext()) != null) {
                          ARawMutation variation = ((ATsvCreator)
objectCreator).evaluate(nextLine);
                          if (variation != null) {
                                 variations.add(variation);
                                 counterSuccessful++;
                          }
                    }

             return variations;
      }
}
```

AObjectCreator.java

```
public abstract class ATsvCreator extends AObjectCreator {

      public abstract ARawMutation evaluate(String[] line);
}
```

ARawMutation.java

```
public abstract class ARawMutation {

      public String id_gene;
      public String id_data_bank;
      protected Container cache = Container.getContainer();

      /**
       * This generic method is used to invoke a translation from any type
of
       * mutation to a variation that corresponds to the Conceptual Schema
of the
       * Human Genome
       *
       * @return an instance of a Variation (might be a Precise, Imprecise,
Indel,
       *         Insertion, Deletion or Inversion
       * @throws IOException
       * @throws FileNotFoundException
```

```
         * @throws SQLException
         */
        public abstract void translate(List<Variation> variations, int iniCDS)
throws FileNotFoundException, IOException, SQLException;

        public abstract specialization_effect getSpecializationEffect();

        public abstract specialization_localization
getSpecializationLocalization();

        /**
         * returns the databank object corresponding to this ARawMutation
         * @return
         */
        public Data_bank getDatabank() {
              return cache.returnDatabank(id_data_bank);
        }

        public abstract specialization_mutant getSpecializationMutant();

        public abstract String getId_variation_db();

        public abstract Bibliography_reference getBibliography_reference();

        public abstract Phenotype getPhenotype();

        /**
         * this method delivers a position that is aligned to the reference
sequence
         * currently used in the Human Genome DataBase
         *
         * @return the position of the mutation aligned to the reference
sequence
         */
        public abstract int getPosition();

        public abstract int getAlignedPosition();

        public abstract String getFlanking_right();

        public abstract String getFlanking_left();

        /**
         * this method delivers the inserted sequence (in the case of HGMD
         * Missense/Nonsense this is usually just one nucleotide)
         *
         * @return the inserted sequence (i.e. in the case of HGMD
Missense/Nonsense
         *         a single nucleotide)
         */
        public abstract String getIns_sequence();

        public abstract String getInsertedSequence();

        public abstract int getIns_repetition();

        public abstract int getInsertedBases();

        public abstract int getDeletedBases();
```

```
	/**
	 * this method tells us how many nucleotides have been deleted from
the
	 * original HGMD codon, compared to the changed codon.
	 *
	 * @return an Integer that determines the amount of deleted bases
	 */
	public abstract int getNum_bases();

	public abstract Boolean isSnp();
}
```

AMutationTransformer.java

```
public abstract class AMutationTransformer {

	protected Map<String, String> parameters;

	@Override
	public boolean execute(Map<String, String> params) throws IOException,
SQLException {
		String geneId = params.get("gene").toUpperCase();

		System.out.println("[INFO]		processing gene " +
geneId);

		this.parameters.putAll(params);

		try {
			List<ARawMutation> input = new ArrayList<ARawMutation>();
			List<Variation> output = new ArrayList<Variation>();

			System.out.println("[INFO]		caching mutations
from input files...");

			AObjectCreator oc = (AObjectCreator)
Class.forName(objectCreator).newInstance();
			input = loadFile(oc, geneId);

			System.out.println("[INFO]		lines succesfully
cached:	" + input.size());
			System.out.println("-------------------------------------
-------------------------------------");
			System.out.println("[INFO]		translating
mutations from cache...");

			int iniCDS = cache.returnStartCDSTranscript(geneId);

			for (ARawMutation bm : input) {
				bm.translate(output, iniCDS);
			}
```

```
                    System.out.println("[INFO]            amount of mutations
successfully translated: " + output.size());

                    List<Variation> uniques = removeDuplicates(output);

                    System.out.println("[INFO]            amount of unique
mutations successfully translated: " + uniques.size());


                    serialize(uniques, workdir, geneId);
              } catch (ClassNotFoundException e) {
                    e.printStackTrace();
                    return false;
              } catch (InstantiationException e) {
                    e.printStackTrace();
                    return false;
              } catch (IllegalAccessException e) {
                    e.printStackTrace();
                    return false;
              }
              return true;
       }

       @Override
       public OptionGroup getOptionsGroup() {
              OptionGroup og = new OptionGroup();
              og.addOption(new Option("gene", "gene", true, "the gene to be
transformed"));
              return og;
       }
```

LoadCore.java

```
public void LoadCore() throws FileNotFoundException, IOException,
SQLException
       {
              String idNG= "";
              String idSymbol="";
              int id_HUGO=0;
              String officialName="";
              String summary="";
              int chr=0;
              String locus="";
              String version="";

              int startGene=0;
              int endGene=0;
              String strand="P";
              String sequence="";

              int startTranscript=0;
              int endTranscript=0;
              int startCDS=0;
              int endCDS=0;
              String isReference="Y";
```

```
            String nm_identifier="";

            String[] exons = new String[100];
            ArrayList<String> allExons = new ArrayList<String>();
            String[] exonsSplit = new String[100];
            String[] allExonsSplit = new String[100];
            int startExon=0;
            int endExon=0;

            String np_identifier="";

            Config config = new Config();
            String dir = System.getProperty("user.dir");
            config.append(new
File(System.getProperty("user.dir")+"/src/main/java/app.properties"));

            HgdbApi acc = new HgdbApi(config.getProperty("DB_HOST"),
Integer.parseInt(config.getProperty("DB_PORT")),
config.getProperty("DB_SID"), config.getProperty("DB_USER"),
config.getProperty("DB_PASSWORD"));
            Connection conn = acc.makeConnection();
            Statement stmt = conn.createStatement();

        Container cache = Container.getContainer();

        String NG;

            String file=System.getProperty("user.home")+"/data";

            TextFileReader tfr = new TextFileReader(file+"/ngGene.txt");
            String line = tfr.readLine();
            String[] fields;
            while (line != null){

                  fields = line.split("\t");
                  idSymbol= fields[0];
                  NG= fields[1];

                  try
              {
                  EUtilsServiceStub service = new EUtilsServiceStub();
                  EUtilsServiceStub.ESearchRequest esearch = new
ESearchRequest();
                  esearch.setDb("nuccore");
                  esearch.setTerm(NG);
                  EUtilsServiceStub.ESearchResult res =
service.run_eSearch(esearch);

                  for(int i = 0; i < res.getIdList().getId().length; i++)
                  {
                        idNG=res.getIdList().getId()[i];
                        System.out.println("idNG:
"+res.getIdList().getId()[i]);
                  }
              }
              catch (Exception e) { System.out.println(e.toString()); }

              try
              {
```

```
                    EFetchSequenceServiceStub service = new
EFetchSequenceServiceStub();

                    EFetchSequenceServiceStub.EFetchRequest req = new
EFetchSequenceServiceStub.EFetchRequest();
                    req.setDb("nuccore");
                    req.setId(idNG);
                    EFetchSequenceServiceStub.EFetchResult res =
service.run_eFetch(req);

                    for (int i = 0; i <
res.getGBSet().getGBSetSequence().length; i++)
                    {
                        EFetchSequenceServiceStub.GBSeq_type0 obj =
res.getGBSet().getGBSetSequence()[i].getGBSeq();
                        System.out.println("Locus: " + obj.getGBSeq_locus());
                        officialName=obj.getGBSeq_definition().substring(0,
obj.getGBSeq_definition().indexOf("(")-1);
                        System.out.println(officialName);
                        System.out.println("Version: " +
obj.getGBSeq_accessionVersion());

summary=obj.getGBSeq_comment().substring(obj.getGBSeq_comment().indexOf("Summ
ary")+8);
                        System.out.println("Comment: " +
obj.getGBSeq_comment().substring(obj.getGBSeq_comment().indexOf("Summary")+8)
);
                        System.out.println("Sequence: " +
obj.getGBSeq_sequence());
                        sequence=obj.getGBSeq_sequence().toUpperCase();

                        strand="P";
                        for(int j = 0; j <
obj.getGBSeq_featureTable().getGBSeq_featureTableSequence().length; j++)
                        {
                            EFetchSequenceServiceStub.GBFeature_type0 fea =
obj.getGBSeq_featureTable().getGBSeq_featureTableSequence()[j].getGBFeature()
;
                            if(fea.getGBFeature_key().equals("source"))
                            {
                                  for(int x = 0; x <
fea.getGBFeature_quals().getGBFeature_qualsSequence().length; x++)
                                 {

      EFetchSequenceServiceStub.GBQualifier_type0 feai =
fea.getGBFeature_quals().getGBFeature_qualsSequence()[x].getGBQualifier();

      if(feai.getGBQualifier_name().equals("map"))
                                        {
                                              locus
=feai.getGBQualifier_value();
                                              System.out.println(locus);
                                        }

      if(feai.getGBQualifier_name().equals("chromosome"))
                                        {
                                              chr
=Integer.parseInt(feai.getGBQualifier_value());
                                              System.out.println(chr);
```

```
                                        }
                                    }
                                }

        if(fea.getGBFeature_key().equals("gene")&&!(fea.getGBFeature_location(
).startsWith("complement")))
                                {
                                        System.out.println("stop");

                                        for(int x = 0; x <
fea.getGBFeature_quals().getGBFeature_qualsSequence().length; x++)
                                    {

        EFetchSequenceServiceStub.GBQualifier_type0 feai =
fea.getGBFeature_quals().getGBFeature_qualsSequence()[x].getGBQualifier();

        if(feai.getGBQualifier_name().equals("gene")&&feai.getGBQualifier_valu
e().equals(idSymbol))
                                        {

        startGene=Integer.parseInt(fea.getGBFeature_location().substring(0,fea
.getGBFeature_location().indexOf("..")));
                                        endGene=
Integer.parseInt(fea.getGBFeature_location().substring(fea.getGBFeature_locat
ion().indexOf("..")+2));
                                        }
                                    }
                                }

        if(fea.getGBFeature_key().equals("mRNA")&&fea.getGBFeature_location().
startsWith("join"))
                                {

        System.out.println(fea.getGBFeature_location());

        startTranscript=Integer.parseInt(fea.getGBFeature_location().substring
(fea.getGBFeature_location().indexOf("(")+1,
fea.getGBFeature_location().indexOf("..")));
                                    endTranscript=
Integer.parseInt(fea.getGBFeature_location().substring(fea.getGBFeature_locat
ion().lastIndexOf("..")+2, fea.getGBFeature_location().indexOf(")")));
                                    for(int x = 0; x <
fea.getGBFeature_quals().getGBFeature_qualsSequence().length; x++)
                                    {

        EFetchSequenceServiceStub.GBQualifier_type0 feai =
fea.getGBFeature_quals().getGBFeature_qualsSequence()[x].getGBQualifier();
                                        System.out.println("Name Qualifier: "
+ feai.getGBQualifier_name());
                                        System.out.println("Value Qualifier: "
+ feai.getGBQualifier_value());

        if(feai.getGBQualifier_name().equals("transcript_id"))
                                        {

        nm_identifier=feai.getGBQualifier_value();
                                                System.out.println("Value
Qualifier: " + feai.getGBQualifier_value());
                                        }
```

```
                                }
                                  exons=fea.getGBFeature_location().split(",");
                            }

      if(fea.getGBFeature_key().equals("CDS")&&fea.getGBFeature_location().s
tartsWith("join"))
                            {

      System.out.println(fea.getGBFeature_location());

      startCDS=Integer.parseInt(fea.getGBFeature_location().substring(fea.ge
tGBFeature_location().indexOf("(")+1,
fea.getGBFeature_location().indexOf("..")));
                                  endCDS=
Integer.parseInt(fea.getGBFeature_location().substring(fea.getGBFeature_locat
ion().lastIndexOf("..")+2, fea.getGBFeature_location().indexOf(")")));

                                  for(int x = 0; x <
fea.getGBFeature_quals().getGBFeature_qualsSequence().length; x++)
                                {

      EFetchSequenceServiceStub.GBQualifier_type0 feai =
fea.getGBFeature_quals().getGBFeature_qualsSequence()[x].getGBQualifier();

      if(feai.getGBQualifier_name().equals("protein_id"))
                                        {

      np_identifier=feai.getGBQualifier_value();
                                              System.out.println("Value
Qualifier: " + feai.getGBQualifier_value());
                                        }

      if(feai.getGBQualifier_name().equals("db_xref")&&feai.getGBQualifier_v
alue().contains("HGNC"))
                                        {
                                              id_HUGO
=Integer.parseInt(feai.getGBQualifier_value().substring(feai.getGBQualifier_v
alue().indexOf(":")+1));

      System.out.println("idHUGO"+id_HUGO);
                                        }
                                }


                            }
                            if(fea.getGBFeature_key().equals("exon"))
                            {
                                  allExons.add(fea.getGBFeature_location());
                            }

                        }
                        System.out.println("----------------------------------
--------");
                    }
                }
                catch (Exception e) { System.out.println(e.toString()); }
```

```
              acc.insertGene(conn, idSymbol, id_HUGO, officialName, summary,
chr, locus);
             acc.insertAllelicReferenceType(conn, idSymbol, startGene,
endGene, strand, sequence, "NCBI", version, idNG);

             ResultSet rsetFindAllele;
             rsetFindAllele = stmt.executeQuery("select allele_num from
allele where id_gene='"+idSymbol+"'");
             boolean allele = rsetFindAllele.next();
             int idAllele = rsetFindAllele.getInt(1);
             if (allele)
             {
                    acc.insertTranscript(conn, idAllele, startTranscript,
endTranscript, startCDS, endCDS, isReference, nm_identifier, np_identifier);
             }
             rsetFindAllele.close();

             ResultSet rsetFindTranscript;
             rsetFindTranscript = stmt.executeQuery("select id_transcript
from transcript where allele_num='"+idAllele+"'");
             rsetFindTranscript.next();
             int idTranscript = rsetFindTranscript.getInt(1);
             rsetFindTranscript.close();

             for(String e : allExons)
             {
                    allExonsSplit=e.split("\\.\\.");
             startExon=Integer.parseInt(allExonsSplit[0]);
             endExon=Integer.parseInt(allExonsSplit[1]);
             acc.insertExon(conn, startExon, endExon);

             }
             allExons=new ArrayList<String>();
             allExonsSplit=new String[100];

             for (int e=0; e<exons.length; e++)
             {
                    System.out.println(e+": "+exons[e].toString());
                    exonsSplit=exons[e].split("\\.\\.");

                    if(exonsSplit[0].startsWith("join"))

       startExon=Integer.parseInt(exonsSplit[0].substring(5));
                    else startExon=Integer.parseInt(exonsSplit[0]);

                    if(exonsSplit[1].endsWith(")"))
                           endExon=
Integer.parseInt(exonsSplit[1].substring(0,exonsSplit[1].length()-1));
                    else endExon=Integer.parseInt(exonsSplit[1]);

                    ResultSet rsetFindExon;
                    rsetFindExon = stmt.executeQuery("select id_exon from exon
where start_exon="+startExon+" and end_exon="+endExon+"");
                    rsetFindExon.next();
                    int idExon = rsetFindExon.getInt(1);
                    rsetFindExon.close();

                    ResultSet rsetInsertExonTranscript;
```

```
                    rsetInsertExonTranscript = stmt.executeQuery("insert into
exon_transcript (id_transcript, id_exon, name) values
("+idTranscript+","+idExon+",1)");
                    rsetInsertExonTranscript.close();
              }

              //ORDER
                line = tfr.readLine();
              }
              stmt.close();
          acc.close(conn);
          conn.close();
          ordenExon();

              }
```

Variation.java

```
public abstract class Variation extends CsObject {

       private static final long serialVersionUID = 1L;

       public enum specialization_effect {
              MUTANT, NEUTRAL, UNKNOWN
       }

       public enum specialization_mutant {
              SPLICING, REGULATORY, MISSENSE, OTHERS
       }

       public enum specialization_localization {
              GENIC, CHROMOSOMIC
       }

       public specialization_effect effect;
       public specialization_mutant mutant;
       public specialization_localization localization;

       public Phenotype phenotype;
       public int id_variation;
       public int allele_num_rt;
       public Data_bank data_bank;
       public String id_variation_db;
       public Bibliography_reference bibliography_reference;
       //multiple bibliography
       public List<Bibliography_reference> bibliography_reference_list;
       public String clinicallySignificance;

       public Variation(specialization_effect effect, specialization_mutant
mutant, specialization_localization localization, int id_variation, int
allele_num_rt,
                    Data_bank data_bank, String id_variation_db,
Bibliography_reference bibliography_reference, String clinicallySignificance)
{
              super();
              this.effect = effect;
```

```
            this.mutant = mutant;
            this.localization = localization;
            this.id_variation = id_variation;
            this.allele_num_rt = allele_num_rt;
            this.data_bank = data_bank;
            this.id_variation_db = id_variation_db;
            this.bibliography_reference = bibliography_reference;
            this.clinicallySignificance=clinicallySignificance;
      }

      public Variation(specialization_effect effect, specialization_mutant
mutant, specialization_localization localization, Data_bank data_bank,
                  int allele_num_rt, String id_variation_db,
Bibliography_reference bibliography_reference, String clinicallySignificance)
{
            super();
            this.effect = effect;
            this.mutant = mutant;
            this.localization = localization;
            this.allele_num_rt = allele_num_rt;
            this.data_bank = data_bank;
            this.id_variation_db = id_variation_db;
            this.bibliography_reference = bibliography_reference;
            this.clinicallySignificance=clinicallySignificance;
      }


      //multiple bibliography

      public Variation(specialization_effect effect, specialization_mutant
mutant, specialization_localization localization, Data_bank data_bank,

                  int allele_num_rt, String id_variation_db,
List<Bibliography_reference> bibliography_reference, String
clinicallySignificance) {
            super();
            this.effect = effect;
            this.mutant = mutant;
            this.localization = localization;
            this.allele_num_rt = allele_num_rt;
            this.data_bank = data_bank;
            this.id_variation_db = id_variation_db;
            //this.bibliography_reference = bibliography_reference;
            this.bibliography_reference_list = bibliography_reference;
            this.clinicallySignificance=clinicallySignificance;
      }

      public Variation(specialization_effect effect, specialization_mutant
mutant, specialization_localization localization, Data_bank data_bank,
                  String id_variation_db, Bibliography_reference
bibliography_reference, String clinicallySignificance) {
            super();
            this.effect = effect;
            this.mutant = mutant;
            this.localization = localization;
            this.data_bank = data_bank;
            this.id_variation_db = id_variation_db;
            this.bibliography_reference = bibliography_reference;
            this.clinicallySignificance=clinicallySignificance;
```

```
	}

	/**
	 * retrieves a unique key for each Variation, usually this will
consist of a
	 * tuple of the following form:
	 *
	 * KEY: <TYPE><POSITION><CHANGE>
	 *
	 * - TYPE can be either deletion, insertion, indel or inversion -
POSITION
	 * position of the variation - CHANGE can be number of deleted bases,
	 * inserted number of times repeated or a combination of all
	 *
	 * @return the Variations' unique key as a String
	 */
	public abstract String getKey();

	public static PreparedStatement getPreparedStatement(Connection conn)
throws SQLException {
		String sql = "INSERT INTO variation(id_variation,
id_allele_num_rt, specialization_effect, specialization_mutant,
specialization_localization, id_data_bank, id_variation_db,
clinically_important) VALUES(?,?,?,?,?,?,?,?)";
		return conn.prepareStatement(sql);
	}

	public String getSpecialization_effect() {
		Map<specialization_effect, String> effects = new
HashMap<specialization_effect, String>();
		effects.put(specialization_effect.MUTANT, "M");
		effects.put(specialization_effect.NEUTRAL, "N");
		effects.put(specialization_effect.UNKNOWN, "U");
		return effects.get(effect);
	}

	public String getSpecialization_mutant() {
		Map<specialization_mutant, String> effects = new
HashMap<specialization_mutant, String>();
		effects.put(specialization_mutant.MISSENSE, "M");
		effects.put(specialization_mutant.OTHERS, "O");
		effects.put(specialization_mutant.REGULATORY, "R");
		effects.put(specialization_mutant.SPLICING, "S");
		return effects.get(mutant);
	}

	public String getSpecialization_localization() {
		Map<specialization_localization, String> effects = new
HashMap<specialization_localization, String>();
		effects.put(specialization_localization.GENIC, "G");
		effects.put(specialization_localization.CHROMOSOMIC, "C");
		return effects.get(localization);
	}

	public String toLine() {
		return "Variation [effect=" + effect + ", mutant=" + mutant + ",
localization=" + localization + ", fenotype=" + phenotype + ", id_variation="
+ id_variation
```

```
                        + ", allele_num_rt=" + allele_num_rt + ",
id_data_bank=" + data_bank.name + ", id_variation_db=" + id_variation_db + ",
publication=" + bibliography_reference + ",
clinicalSignificance="+clinicallySignificance+"]";
	}
}
```

Precise.java

```
public class Precise extends Variation {

	private static final long serialVersionUID = 1L;

	public int position;
	public String flanking_left;
	public String flanking_right;
	public Type type;
	public String ins_sequence;
	public int ins_repetition;
	public int num_bases;
	public Boolean snp;
	public String localizationType;
	public String alnQuality;

	public enum Type {
		INSERTION, DELETION, INDEL, INVERSION
	}

	public Precise(specialization_effect effect, specialization_mutant
mutant, specialization_localization localization, int id_variation, int
allele_num_rt,
			Data_bank data_bank, String id_variation_db,
Bibliography_reference publication, String clinicallySignificance, Phenotype
fenotype, int position, String flanking_left,
			String flanking_right, Type type, String ins_sequence, int
ins_repetition, int num_bases, Boolean snp, String localizationType, String
alnQuality) {
		super(effect, mutant, localization, id_variation, allele_num_rt,
data_bank, id_variation_db, publication, clinicallySignificance);
		this.phenotype = fenotype;
		this.position = position;
		this.flanking_left = flanking_left;
		this.flanking_right = flanking_right;
		this.type = type;
		this.ins_sequence = ins_sequence;
		this.ins_repetition = ins_repetition;
		this.num_bases = num_bases;
		this.snp = snp;
		this.localizationType=localizationType;
		this.alnQuality=alnQuality;
	}

	public Precise(specialization_effect effect, specialization_mutant
mutant, specialization_localization localization, Data_bank data_bank, String
id_variation_db,
```

```
                    Bibliography_reference publication, String
clinicallySignificance, Phenotype phenotype, int position, String
flanking_left, String flanking_right, Type type, String ins_sequence,
                    int ins_repetition, int num_bases, Boolean snp, String
localizationType, String alnQuality) {
             super(effect, mutant, localization, data_bank, id_variation_db,
publication, clinicallySignificance);
             this.phenotype = phenotype;
             this.position = position;
             this.flanking_left = flanking_left;
             this.flanking_right = flanking_right;
             this.type = type;
             this.ins_sequence = ins_sequence;
             this.ins_repetition = ins_repetition;
             this.num_bases = num_bases;
             this.snp = snp;
             this.localizationType=localizationType;
             this.alnQuality=alnQuality;
      }

      public Precise(specialization_effect effect, specialization_mutant
mutant, specialization_localization localization, int allele_num_rt,
Data_bank id_data_bank, String id_variation_db,
                    Bibliography_reference publication, String
clinicallySignificance, Phenotype fenotype, int position, String
flanking_left, String flanking_right, Type type, String ins_sequence, int
ins_repetition, int num_bases, Boolean snp, String localizationType, String
alnQuality)
      {
             super(effect, mutant, localization, id_data_bank, allele_num_rt,
id_variation_db, publication, clinicallySignificance);
             this.phenotype = fenotype;
             this.position = position;
             this.flanking_left = flanking_left;
             this.flanking_right = flanking_right;
             this.type = type;
             this.ins_sequence = ins_sequence;
             this.ins_repetition = ins_repetition;
             this.num_bases = num_bases;
             this.snp = snp;
             this.localizationType=localizationType;
             this.alnQuality=alnQuality;

      }

      //multiple Bibliography
      public Precise(specialization_effect effect, specialization_mutant
mutant, specialization_localization localization, int allele_num_rt,
Data_bank id_data_bank, String id_variation_db,
                    List<Bibliography_reference> publication, String
clinicallySignificance, Phenotype fenotype, int position, String
flanking_left, String flanking_right, Type type, String ins_sequence, int
ins_repetition, int num_bases, Boolean snp, String localizationType, String
alnQuality)
      {
             super(effect, mutant, localization, id_data_bank, allele_num_rt,
id_variation_db, publication, clinicallySignificance);
             this.phenotype = fenotype;
             this.position = position;
```

```
		this.flanking_left = flanking_left;
		this.flanking_right = flanking_right;
		this.type = type;
		this.ins_sequence = ins_sequence;
		this.ins_repetition = ins_repetition;
		this.num_bases = num_bases;
		this.snp = snp;
		this.localizationType=localizationType;
		this.alnQuality=alnQuality;

	}

	@Override
	public String getKey() {
		if (phenotype == null) {
			phenotype = new Phenotype("");
		}

		if (type == Type.DELETION) {
			return "DEL" + position + num_bases + snp + phenotype.name
+ data_bank.id_data_bank;
		} else if (type == Type.INSERTION) {
			return "INS" + position + ins_sequence + snp +
phenotype.name + data_bank.id_data_bank;
		} else if (type == Type.INDEL) {
			return "IND" + position + ins_sequence + num_bases + snp +
phenotype.name + data_bank.id_data_bank;
		}
		return id_variation_db;
	}

	public static PreparedStatement getPreparedStatement(Connection conn)
throws SQLException {
		String sql = "INSERT INTO precise(id_variation, position, type,
ins_sequence, ins_repetition, num_bases, SNP, flanking_left, flanking_right,
alnQuality, localizationType) VALUES(?,?,?,?,?,?,?,?,?,?,?)";
		return conn.prepareStatement(sql);
	}

	public String getType() {
		Map<Type, String> types = new HashMap<Type, String>();
		types.put(Type.INSERTION, "IS");
		types.put(Type.DELETION, "DE");
		types.put(Type.INDEL, "ID");
		types.put(Type.INVERSION, "IV");
		return types.get(type);
	}

	@Override
	public String toString() {

		return "Precise [position=" + position + ", flanking_left=" +
flanking_left + ", flanking_right=" + flanking_right + ", type=" + type + ",
ins_sequence="
				+ ins_sequence + ", ins_repetition=" +
ins_repetition + ", num_bases=" + num_bases + ", snp=" + snp + ",
alnQuality=" + alnQuality+ ", localizationType=" + localizationType+ "]";
	}
```

MutationLoader.java

```
public class MutationLoader extends ALoader {

	@Override
	public boolean execute(Map<String, String> params) throws IOException
{
		purgeTables(); // make sure we start with empty tables

		try {
			PlugoutFactory pf = new PlugoutFactory();
			APlugout plugout =
pf.createPlugOut(params.get("plugout"));

			File varPath = new File(workdir, "clean" + File.separator
+ "variations");
			List<Variation> variations = null;
			File[] files = varPath.listFiles();
			for (File f : files) {
				String fileName = f.getName();
				if (!fileName.startsWith(".")) {
					String gene = fileName.substring(0,
fileName.indexOf("."));
					variations = readVariationsFile(f);
					System.out.println("[INFO]             size
of " + gene + " cache:    " + variations.size());

					try {
						plugout.processVariations(variations,
gene);
					} catch (SQLException e1) {
						e1.printStackTrace();
					}
				}
				System.out.println();
			}
		} catch (InstantiationException e) {
			e.printStackTrace();
		} catch (IllegalAccessException e) {
			e.printStackTrace();
		} catch (ClassNotFoundException e) {
			e.printStackTrace();
		}
		return false;
	}

	private void purgeTables() {
		cache.purgeTable("reference_variation");
		cache.purgeTable("bibliography_db_location");
		cache.purgeTable("bibliography_reference");
		cache.purgeTable("bibliography_databank");
		cache.purgeTable("certainty");
		cache.purgeTable("phenotype");
		cache.purgeTable("precise");
		cache.purgeTable("imprecise");
		cache.purgeTable("variation");
	}
```

```
	@Override
	public boolean initialize(Map<String, String> params) {
		// TODO Auto-generated method stub
		return false;
	}

	@Override
	public OptionGroup getOptionsGroup() {
		OptionGroup og = new OptionGroup();
		og.addOption(new Option("plugout", "plugout", true, "the plugout
to which is going to be output: e.g. SQL, FILE, etc."));
		return og;
	}
```

SqlOutManager.java

```
public class SqlOutManager extends APlugout {

	private int bibrefIndex;
	private int bibdbIndex;
	@Override
	public void processVariations(List<Variation> variations, String gene)
throws SQLException {
		List<Phenotype> phenotypes = new ArrayList<Phenotype>();
		List<Certainty> certainties = new ArrayList<Certainty>();
		List<Data_bank> data_banks = new ArrayList<Data_bank>();

		List<Bibliography_reference> bibliography_references = new
ArrayList<Bibliography_reference>();
		List<Bibliography_data_bank> bibliography_data_banks = new
ArrayList<Bibliography_data_bank>();
		List<Bib_ref_location> bib_ref_locations = new
ArrayList<Bib_ref_location>();
		List<Reference_variation> reference_variations = new
ArrayList<Reference_variation>();

		List<Precise> precises = new ArrayList<Precise>();
		List<Imprecise> imprecises = new ArrayList<Imprecise>();

		Certainty cert = null;
		int phenIndex = cache.getLastIndex("id_phenotype", "phenotype");
		int variationIndex = cache.getLastIndex("id_variation",
"variation");
		 bibrefIndex = cache.getLastIndex("id_bib_ref",
"bibliography_reference");
		 bibdbIndex = cache.getLastIndex("id_bib_db",
"bibliography_databank");

		for (Variation var : variations) {
			var.id_variation = variationIndex;
			if(var.id_variation_db.equals("APCc.382A>T"))
			{
				System.out.println("Variacion con multiples
referencias");
			}
```

```
                    var.allele_num_rt =
cache.returnGene(gene).getReferenceAllele("NCBI").allele_num;

                    if (var.phenotype != null) {
                          var.phenotype.id_phenotype = phenIndex;

                          Phenotype phen =
cache.returnPhenotype(var.phenotype.name);
                          if (phen == null) {
                                phen = var.phenotype;
                                if (!phenotypes.contains(phen)) {
                                      phenotypes.add(phen);
                                }
                          } else {
                                var.phenotype = phen;
                          }
                          if(!var.data_bank.name.equals("HGMD"))
                          {
                                cert = new Certainty(var.phenotype, var, "");
                          }
                          else
                          {
                                if(var.phenotype.name.equals(""))
                                {
                                      cert = new Certainty(var.phenotype,
var, "");
                                }

                                else
                                {
                                      if(var.phenotype.name.contains("?"))
                                      {
                                            cert = new
Certainty(var.phenotype, var, "D");
                                      }
                                      else
                                      {
                                            cert = new
Certainty(var.phenotype, var, "S");
                                      }
                                }
                          }

                          certainties.add(cert);

                          phenIndex++;
                    }

                    Data_bank db =
cache.returnDatabank(var.data_bank.id_data_bank);
                    if (db == null) {
                          db = var.data_bank;
                          if (!data_banks.contains(db)) {
                                data_banks.add(db);
                          }
                    } else {
                          var.data_bank = db;
                    }
```

```
// ====================== MULTIPLE REFERENCES INSERT ======================

                    if (var.bibliography_reference != null ||
var.bibliography_reference_list != null) {
                          if(var.bibliography_reference_list==null) {

      oneVariationRef(var,bibrefIndex,bibdbIndex,bibliography_references,ref
erence_variations,bib_ref_locations,bibliography_data_banks);
                          }
                          else{

      multipleVariationRef(var,bibrefIndex,bibdbIndex,bibliography_reference
s,reference_variations,bib_ref_locations,bibliography_data_banks);
                          }

                    if (var instanceof Precise) {
                          precises.add((Precise) var);
                    }
                    if (var instanceof Imprecise) {
                          imprecises.add((Imprecise) var);
                    }

                    variationIndex++;
                    bibrefIndex++;
                    bibdbIndex++;
              }

              cache.phenotypeCache.clear();
              cache.bibliography_data_bankCache.clear();
              cache.data_bankCache.clear();

              insertData_banks(data_banks);
              insertVariations(variations, gene);
              insertPrecises(precises);
              insertImprecises(imprecises);
              insertPhenotypes(phenotypes);
              insertCertainties(certainties);
              insertBibliography_references(bibliography_references);
              insertBibliography_data_banks(bibliography_data_banks);
              insertBib_ref_locations(bib_ref_locations);
              insertReference_variations(reference_variations);
      }


      public void oneVariationRef(Variation var, int bibrefIndex,int
bibdbIndex, List<Bibliography_reference> bibliography_references,
                          List<Reference_variation>
reference_variations,List<Bib_ref_location>
bib_ref_locations,List<Bibliography_data_bank> bibliography_data_banks){
      var.bibliography_reference.bib_ref = bibrefIndex;
            var.bibliography_reference.bibliography_data_bank.id_bib_db =
bibdbIndex;

            bibliography_references.add(var.bibliography_reference);

            Bibliography_data_bank bib_db =
cache.returnBibliography_data_bank(var.bibliography_reference.bibliography_da
ta_bank.bibliography_db_name);
            if (bib_db == null) {
```

```
                  bib_db =
var.bibliography_reference.bibliography_data_bank;
                  if (!bibliography_data_banks.contains(bib_db)) {
                        bibliography_data_banks.add(bib_db);
                  }
            } else {
                  var.bibliography_reference.bibliography_data_bank =
bib_db;
            }

            reference_variations.add(new Reference_variation(var,
var.bibliography_reference));

            if(var.bibliography_reference.bibliography_data_bank==null ||
var.bibliography_reference==null) {
                  System.err.println();
            }

            bib_ref_locations
                        .add(new
Bib_ref_location(var.bibliography_reference.bibliography_data_bank,
var.bibliography_reference, var.bibliography_reference.url));
      }



      // MULTIPLE REFERENCES
      public void multipleVariationRef(Variation var, int bibrefIndex,int
bibdbIndex, List<Bibliography_reference> bibliography_references,
                  List<Reference_variation>
reference_variations,List<Bib_ref_location>
bib_ref_locations,List<Bibliography_data_bank> bibliography_data_banks){
            int bibrefIndexcopy = bibrefIndex;
            int bibdbIndexcopy = bibdbIndex;
            for(int i=0;i<var.bibliography_reference_list.size();i++){
                  var.bibliography_reference_list.get(i).bib_ref =
bibrefIndexcopy;

      var.bibliography_reference_list.get(i).bibliography_data_bank.id_bib_d
b = bibdbIndexcopy;

      bibliography_references.add(var.bibliography_reference_list.get(i));


      Bibliography_data_bank bib_db =
cache.returnBibliography_data_bank(var.bibliography_reference_list.get(i).bib
liography_data_bank.bibliography_db_name);
      if (bib_db == null) {
            bib_db =
var.bibliography_reference_list.get(i).bibliography_data_bank;
            if (!bibliography_data_banks.contains(bib_db)) {
                  bibliography_data_banks.add(bib_db);
            }
      } else {
            var.bibliography_reference_list.get(i).bibliography_data_bank =
bib_db;
      }
```

```
		reference_variations.add(new Reference_variation(var,
var.bibliography_reference_list.get(i)));

		if(var.bibliography_reference_list.get(i).bibliography_data_bank==null
|| var.bibliography_reference_list.get(i)==null) {
			System.err.println();
		}

		bib_ref_locations
				.add(new
Bib_ref_location(var.bibliography_reference_list.get(i).bibliography_data_ban
k, var.bibliography_reference_list.get(i),
var.bibliography_reference_list.get(i).url));

		if(var.bibliography_reference_list.size() - i != 1){
			bibrefIndexcopy++;
			bibdbIndexcopy++;
		}
		}//for

			//asign index to variables
			this.bibdbIndex = bibdbIndexcopy;
			this.bibrefIndex = bibrefIndexcopy;

}

		public void insertVariations(List<Variation> variations, String gene)
throws SQLException {
			PreparedStatement ps =
Variation.getPreparedStatement(cache.conn);

			for (Variation var : variations) {
				ps.setInt(1, var.id_variation);
				ps.setInt(2, var.allele_num_rt);
				ps.setString(3, var.getSpecialization_effect());
				ps.setString(4, var.getSpecialization_mutant());
				ps.setString(5, var.getSpecialization_localization());
				ps.setString(6, var.data_bank.id_data_bank);
				ps.setString(7, var.id_variation_db);
				ps.setString(8, var.clinicallySignificance);
				ps.addBatch();
			}

			ps.executeBatch();
			ps.close();
		}

		@Override
		public void insertPrecises(List<Precise> precises) throws SQLException
{
			PreparedStatement ps = Precise.getPreparedStatement(cache.conn);

			for (Precise prec : precises) {
				if (prec.ins_sequence == null ||
prec.ins_sequence.length() < 4000) {
					ps.setInt(1, prec.id_variation);
					ps.setInt(2, prec.position);
					ps.setString(3, prec.getType());
					if(prec.ins_sequence==null)
```

```
                                    ps.setString(4, null);
                              else
                                    ps.setString(4,
prec.ins_sequence.toUpperCase());
                              ps.setInt(5, prec.ins_repetition);
                              ps.setInt(6, prec.num_bases);
                              if(prec.snp==null)
                                    ps.setString(7, null);
                              else
                                    ps.setBoolean(7, prec.snp);
                              ps.setString(8, prec.flanking_left);
                              ps.setString(9, prec.flanking_right);
                              ps.setString(10, prec.alnQuality);
                              ps.setString(11, prec.localizationType);
                              ps.addBatch();
                        } else {
                              System.err.println("[WARNING]   the length of this
particular mutations' inserted sequence is larger then 4000 characters, can
not be inserted in DB therefore");
                        }
                  }

                  ps.executeBatch();
                  ps.close();

            }

            @Override
            public void insertImprecises(List<Imprecise> imprecises) throws
SQLException {
                  PreparedStatement ps =
Imprecise.getPreparedStatement(cache.conn);

                  for (Imprecise imp : imprecises) {
                        ps.setInt(1, imp.id_variation);
                        ps.setString(2, imp.description);
                        ps.setString(3, imp.getType());
                        ps.addBatch();
                  }

                  ps.executeBatch();
                  ps.close();
            }

            @Override
            public void insertBibliography_references(List<Bibliography_reference>
publications) throws SQLException {
                  PreparedStatement ps =
Bibliography_reference.getPreparedStatement(cache.conn);

                  for (Bibliography_reference bibref : publications) {
                        ps.setInt(1, bibref.bib_ref);
                        ps.setString(2, bibref.title);
                        ps.setString(3, bibref.summary);
                        ps.setString(4, bibref.publication);
                        ps.setString(5, bibref.authors);
                        ps.setDate(6, bibref.date_pub);
                        ps.addBatch();
                        }
```

```
			ps.executeBatch();
			ps.close();
		}

		public void insertBib_ref_locations(List<Bib_ref_location>
bib_ref_locations) throws SQLException {
			PreparedStatement ps =
Bib_ref_location.getPreparedStatement(cache.conn);

			for (Bib_ref_location b : bib_ref_locations) {
				ps.setInt(1, b.bibliography_reference.bib_ref);
				ps.setInt(2, b.bibliography_databank.id_bib_db);
				ps.setString(3, b.url.toString());
				ps.addBatch();
			}

			ps.executeBatch();
			ps.close();
		}

		public void insertBibliography_data_banks(List<Bibliography_data_bank>
bibliography_data_banks) throws SQLException {
			PreparedStatement ps =
Bibliography_data_bank.getPreparedStatement(cache.conn);

			for (Bibliography_data_bank b : bibliography_data_banks) {
				ps.setInt(1, b.id_bib_db);
				ps.setString(2, b.bibliography_db_name);
				ps.addBatch();
			}

			ps.executeBatch();
			ps.close();
		}

		public void insertReference_variations(List<Reference_variation>
reference_variations) throws SQLException {
			PreparedStatement ps =
Reference_variation.getPreparedStatement(cache.conn);

			for (Reference_variation r : reference_variations) {
				ps.setInt(1, r.variation.id_variation);
				ps.setInt(2, r.bibliography_reference.bib_ref);
				ps.addBatch();
			}

			ps.executeBatch();
			ps.close();
		}

		@Override
		public void insertPhenotypes(List<Phenotype> phenotypes) throws
SQLException {
			PreparedStatement ps =
Phenotype.getPreparedStatement(cache.conn);

			for (Phenotype p : phenotypes) {
				ps.setInt(1, p.id_phenotype);
```

```
                ps.setString(2, p.name);
                ps.addBatch();
            }

            ps.executeBatch();
            ps.close();
        }

        public void insertCertainties(List<Certainty> certainties) throws
SQLException {
            PreparedStatement ps =
Certainty.getPreparedStatement(cache.conn);

            for (Certainty c : certainties) {
                ps.setInt(1, c.phenotype.id_phenotype);
                ps.setInt(2, c.var.id_variation);
                ps.setString(3, c.level_certainty);
                ps.addBatch();
            }

            ps.executeBatch();
            ps.close();
        }

        @Override
        public void insertGenes(List<Gene> genes) throws SQLException {
            // TODO Auto-generated method stub

        }

        @Override
        public void insertAlleles(List<Allele> alleles) throws SQLException {
            // TODO revise this method
            PreparedStatement ps = Allele.getPreparedStatement(cache.conn);

            for (Allele allele : alleles) {
                ps.setInt(1, allele.allele_num);
                ps.setInt(2, allele.start_position);
                ps.setInt(3, allele.end_position);
                ps.setString(4, allele.getStrand());
                ps.setString(5, allele.gene.id_symbol);
                ps.addBatch();
            }

            ps.executeBatch();
            ps.close();
        }

        @Override
        public void insertData_banks(List<Data_bank> data_banks) throws
SQLException {
            PreparedStatement ps =
Data_bank.getPreparedStatement(cache.conn);

            for (Data_bank d : data_banks) {
                ps.setString(1, d.id_data_bank);
                ps.setString(2, d.name);
                ps.setString(3, d.description);
                ps.addBatch();
```

```
        }

        ps.executeBatch();
        ps.close();
}
}
```