UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Gestión de Datos Genómicos basada en Modelos Conceptuales

TRABAJO FIN DE MÁSTER

Máster Universitario en Ingeniería Informática

Autor: Alberto García Simón

Tutor: Juan Carlos Casamayor Ródenas

Cotutor: Oscar Pastor López

Curso 2017-2018

Resum

Aquest TFM abordará la problemàtica associada a la gestió de dades genòmiques. Es tracta d'un context de treball especialment complex a causa del volum, heterogeneïtat i dispersió en les fonts de dades existents. El domini d'estudi serà el genoma dels cítrics (gestió de dades genòmiques per als cítrics en l'àmbit de l'Institut Valencià d'Investigacions Agràries (IVIA)). Durant el desenvolupament d'aquest treball s'incidirà en la importància de dissenyar un model conceptual que servisca com a artefacte sistèmic de referència amb l'objectiu de caracteritzar les dades rellevants del domini estudiat d'una manera clara i que siga independent de la plataforma *software* a utilitzar.

Un punt essencial associat a la arquitectura d'aquesta plataforma *software* serà la selecció del tipus de Sistema de Gestió de Base de dades més apropiat per a la manipulació de les dades objecte d'estudi. Per aquesta raó, s'analitzarà en particular la conveniència d'usar un Sistema de Gestió de Bases de Dades (SGBD) relacional o un del tipus NoSQL estudiant les fortaleses i debilitats en relació al domini estudiat. Una vegada elegida la plataforma d'enmagatzematge de dades, es disenyarà i desenvoluparà la Base de Dades corresponent, es carregaràn les dades rellevants i s'elaboraràn les consultes que representen els requisits dels usuaris del sistema.

Tot això proporcionarà un ambient avançat de gestió de dades genòmiques, dirigit per un model conceptual, desenvolupat amb tecnologíes d'última generació i posat a disposició dels usuaris de l'entorn de treball seleccionar.

Paraules clau: Genètica, Models, Neo4j, Python, NodeJS

Resumen

El TFM abordará la problemática asociada a la gestión de datos genómicos. Se trata de un contexto de trabajo especialmente complejo debido al volumen, heterogeneidad y dispersión en las fuentes de datos existentes. El dominio de estudio será el genoma de los cítricos (gestión de datos genómicos para los cítricos en el ámbito del Instituto Valenciano de Investigaciones Agrarias (IVIA)). Durante el desarrollo de este trabajo se incidirá en la importancia de diseñar un Modelo Conceptual que sirva como artefacto sistémico de referencia con el objetivo de caracterizar los datos relevantes del dominio estudiado de una manera clara y que sea independiente de la plataforma *software* a utilizar.

Un punto esencial asociado a la arquitectura de dicha plataforma *software* será la selección del tipo de Sistema de Gestión de Base de Datos más apropiado para la manipulación de los datos objeto de estudio. Por ello, se analizará en particular la conveniencia de usar un Sistema de Gestión de Base de Datos (SGBD) relacional o uno del tipo NoSQL estudiando sus fortalezas y debilidades en relación al dominio estudiado. Una vez decidida la plataforma de almacenamiento de los datos, se diseñará y desarrollará la Base de Datos correspondiente, se cargarán los datos relevantes y se elaborarán las consultas que representen los requisitos de los usuarios del Sistema.

Todo ello proporcionará un ambiente avanzado de gestión de datos genómicos, dirigido por un modelo conceptual, desarrollado con tecnologías de última generación, y puesto a disposición de los usuarios del entorno de trabajo seleccionado.

Palabras clave: Genética, Modelos, Neo4j, Python, NodeJS

Abstract

This thesis will address the problem associated to the management of genomic data. It is a particularly complex work context due to the volume, heterogeneity and dispersion present in the existing data source. The study domain will be the genome of the citrus (Citrus genomic data management by the Valencian Institute of Agrarian Research (IVIA) scientists). During the development of this work we will focus on the importance of designing a Conceptual Model that serves as a reference systemic artifact with the objective of characterizing relevant data of the domain studied in a clear way and platform independent.

A crucial point associated to this software platform will be the selection of type of Database Manage System for the manipulation of the data under study. Therefore, a relational vs NoSQL study will be made analyzing the strengths and weaknesses of both types of systems regarding the studied domain. Once the Database Management System (DBMS) has been decided, it will be designed and developed, the data will be loaded and the queries that represent the requirements of the users will be created.

All this will provide an advanced, model-driven genomic data management environment using the latest generation technologies which will be made available to the users of the selected environment.

Key words: Genetics, Models, Neo4j, Python, NodeJS

Índice general

Índice de figuras

Índice de tablas

Índice de Fragmentos de Código

Parte I

Prefacio

CAPÍTULO 1
Introducción

Este primer capítulo está formado por cuatro secciones. En la primera de ellas (sección 1.1) se presenta las razones que han motivado la realización de este proyecto; ¿por qué hacer un proyecto en este ámbito?, ¿qué puede aportar? En la segunda sección (sección 1.2) se encuentran los objetivos definidos para la realización de este Tésis de Fin de Máster (TFM); ¿qué se quiere obtener?, ¿qué conocimientos se espera haber obtenido tras la finalización del mismo? Por último, la tercera sección (sección 1.3) detalla la estructura de la memoria con un breve resumen del contenido de cada capítulo. Con el objetivo de facilitar la lectura del documento se ha dispuesto un glosario tanto de términos como de acrónimos.

1.1 Motivación

El ser humano es curioso por naturaleza. Es esta curiosidad la que nos ha llevado a indagar, desde el principio de los tiempos, cómo funcionan las cosas y qué las mueve. El *cómo*, *cuándo*, *dónde*, *por qué* y el *para qué* de la naturaleza de los objetos y eventos que se encuentran a nuestro alrededor nos ha llevado a preguntarnos desde las cuestiones más triviales hasta conceptos que escapan a nuestro entendimiento. En no pocas ocasiones hemos sido capaces de desarrollar nuevas teorías para explicar aquello que no entendemos.

Una de las áreas que más importancia ha tomado en los últimos tiempos (aunque siempre ha estado presente en nuestro pensamiento) es el entendimiento de la vida; de qué está formada, cómo cambia ... y estas preguntas nos llevan inevitablemente a la biología y, más concretamente, la genética y genómica. Estas áreas nos permiten empezar a desentrañar los misterios ocultos de la vida.

En un inicio se empezó intentando el qué de las cosas. Por ejemplo, qué es el ADN, qué es una proteína, qué es una variación, etc. Han sido el punto de inicio sobre el cual construir nuestra pirámide de conocimiento. Actualmente, se está sustituyendo el tipo de pregunta y se está sustituyendo el qué por el cómo. Este cambio implica un entendimiento mucho más completo y complejo del dominio de estudio. Por ejemplo, disponemos de una máquina de café, el qué implica saber que, si introducimos una cantidad determinada de dinero, obtendremos un café. Es conocimiento, pero es conocimiento limitado. El siguiente paso sería saber cómo se obtiene ese café, los mecanismos internos que permiten que, al

introducir una moneda obtengamos café. Consiste en adentrarnos en esa caja negra donde únicamente obtenemos una salida a partir de una entrada para hacer nuestra dicha caja y poder explicar los procesos internos que se realizan. Dicho de otra forma, desentrañar todos los detalles y procedimientos que compone la máquina de café. Este ejemplo puede extrapolarse al mundo de la genómica. En los inicios se perseguía el qué: qué es un gen, qué es una proteína. Actualmente la pregunta sería cómo afecta la expresión de un gen en el ser humano o cómo interactúan las proteínas entre ellas.

Ha surgido una oportunidad de colaborar y aportar nuestro granito de arena a esta pirámide de conocimiento de manos del Instituto Valenciano de Investigaciones Agrarias (IVIA)[1]. Se trata de un grupo de investigación de referencia mundial en el ámbito genómico-cítrico. Han ralizado trabajos apasionantes relacionados con los cítricos. Uno de sus principales trabajos ha sido su estudio sobre el análisis y evolución de los cítricos [1]. Poder colaborar con un grupo de esta importancia es motivación más que suficiente. El objetivo que persiguen es determinar en qué se diferencian las distintas especies de cítricos entre sí (procesos de secuenciación genética y comparación de estas secuencias) y (después del qué el cómo) cómo afectan estos cambios a las diferentes variedades.

- ¿Por qué una variedad de cítrico es más dulce que otra?
- ¿por qué una variedad presenta su piel de color naranja y otra amarillo?
- ¿por qué una variedad se adapta mejor a un determinado clima que otra?

Estas son algunas de las preguntas que pueden ser respondidas entendiendo el cómo. Estamos ante un área del conocimiento con apasionantes y casi infinitas posibilidades de estudio

1.2 Objetivos

El trabajo se articula alrededor de dos ejes principales que contribuyen a un único objetivo. Dicho objetivo consiste en el desarrollo de una solución que permita, por un lado, agilizar los procesos de análisis genéticos y, por otro lado, mejorar el tratamiento de datos genómicos. Esta necesidad de mejora se enmarca en el ámbito genómico-cítrico de la mano del IVIA.

Como solución para alcanzar estos objetivos se plantea la creación de un esquema conceptual del genoma de los cítricos con el fin de identificar y definir correctamente los elementos participantes en los procesos de análisis y, posteriormente, el desarrollo de una aplicación de explotación de datos genómicos. Esta plataforma debe nutrirse del esquema conceptual definido anteriormente y de los datos genómicos disponibles. La plataforma debe cumplir una serie de requisitos:

- Capacidad de tratamiento masivo de datos.
- Alta escalabilidad.

[1] http://www.ivia.gva.es/

– Simplicidad de uso.

– Eficiencia en el proceso de análisis.

El primer eje consiste en la generación de un esquema conceptual del genoma de los cítricos. Para ello, en primer, lugar deben analizar los ficheros fuente con los que trabaja el IVIA. Estos ficheros provienen de diversas fuentes las cuales presentan un alto grado de heterogeneidad. El proceso de entendimiento del contenido de los datos se debe apoyar en el conocimiento experto de nuestros colaboradores para asegurar una correcta interpretación del contenido y la forma de los datos. Una vez se ha logrado el conocimiento suficiente como para entender el dominio, se ha procedido a la generación del esquema conceptual que proporcione una sólida base ontológica que permita continuar con el trabajo. Este modelo ha sido revisado, modificado y validado por los expertos del IVIA para asegurar su validez, corrección y utilidad. De este eje se desprenden subobjetivos tales como la obtención del conocimiento necesario en el campo de la genómica y la bioinformática o las metodologías de trabajo de los centros de investigación del ámbito genómico-cítrico.

El segundo eje se fundamenta en el desarrollo de un sistema que permita realizar la explotación de los datos de una manera más eficiente. Ha sido necesario, como paso previo, un análisis de las tecnologías actuales para determinar cuáles se adaptan mejor a la necesidades concretas y particularidades del sistema desarrollado. En primer lugar, se ha procedido a implementar el esquema conceptual de los cítricos generado, Modelo Independiente de la Plataforma (PIM), en el modelo de datos seleccionado, Modelo Específico de la Pataforma (PSM). En segundo lugar, se ha implementado una aplicación que permite a los biotecnólogos interactuar con los datos siguiendo el nuevo esquema y realizar sus estudios de una manera mucho más rápida, eficaz y eficiente.

1.3 Estructura de la Memoria

La memoria comprende dos partes: prefacio y desarrollo. La primera parte se centra en cuestiones teóricas mientras que la segunda pone el foco en todo el trabajo realizado. Dentro de esta primera parte encontramos dos capítulos. En el primer capítulo, la introducción, se han incluido los objetivos de este TFM, así como las motivaciones que han inspirado a la realización del mismo. El segundo y último capítulo de la primera parte de la memoria se proporciona el marco metodológico utilizado para el desarrollo del proyecto y la aplicación del mismo. Además, se puede encontrar una breve introducción conceptual al dominio de la genética y la genómica con el objetivo de facilitar la lectura del documento. Finalmente, se muestra el caso de uso en el cual se ha elaborado este TFM.

En la segunda parte del documento se exponen los distintos desarrollos realizados durante el proyecto. Se divide en tres capítulos: diseño de la solución, validación de la solución y conclusiones. En el primer capítulo, diseño de la solución, se identifica el proceso seguido para le generación del Esquema Conceptual del Genoma de los Cítricos (ECGCit.). Para ello, en primer lugar, se analizan las distintas fuentes de datos y los formatos de los mismos para, posteriormente, di-

señar un esquema que sirva de base ontológica sobre la que desarrollar futuras aplicaciones. Se realiza la especificación de requisitos y se diseña y desarrolla un prototipo de aplicación web.

Inicialmente, se ha realizado un estudio de las distintas tecnologías de bases de datos para seleccionar aquella que mejor se adecúe a las características de los datos. Tras la fase de selección del Sistema de Gestión de Bases de Datos (SGBD), se ha aplicado un proceso ETL que ha culminado con la base de datos generada, poblada, configurada, y funcional. A continuación, se ha diseñado la Interfaz de programación de aplicaciones, del inglés *Application Programming Interface* (API) que debía exponer el servicio. Posteriormente, se ha implementado el servidor, encargado de responder a las peticiones implementando la API diseñada anteriormente; este desarrollo se ha desarrollado siguiendo un diseño modular y escalable. Finalmente, se muestra la interfaz de usuario diseñada, así como las posibles interacciones con ella.

En el segundo capítulo se cuenta cómo se ha ejecutado la validación de la solución. Para ello, se han utilizado técnicas de experimento de caso único e investigación técnica entre otros. Este capítulo concluye con el análisis de los resultados obtenidos tras la ejecución de las validaciones definidas.

Por último, en el tercer capítulo de la segunda parte se presentan las conclusiones de este trabajo, exponiendo las lecciones aprendidas, futuras mejoras y funcionalidades a implementar, determinando la futura evolución del prototipo.

CAPÍTULO 2
Investigación del Problema

El capítulo dos está formado por cinco secciones. En la primera de ellas, metodología de investigación (sección 2.1), se introduce la metodología de investigación aplicada al proceso de desarrollo de este TFM. La segunda sección (sección 2.2) introduce los conocimientos básicos necesarios para seguir sin problemas el contenido de este documento. La tercera sección (sección 2.3 presenta el contexto en el cual se ha realizado el TFM introduciendo características y limitaciones del proyecto. Se trata de un punto clave para entender las motivaciones que han llevado a la creación de la plataforma de análisis documentada en este documento. La cuarta sección (sección 2.4) caracteriza los *stakeholders* (interesados) identificando sus objetivos. Finalmente, la quinta y última sección (sección 2.5) muestra las preguntas de investigación generadas una vez entendido el contexto del problema y que han guiado el desarrollo del proyecto.

2.1 Metodología de Investigación

Para la realización de esta investigación se ha seguido la metodología de investigación *Design Science* propuesta por Wieringa[2].

Esta metodología expone que a la hora de realizar un proyecto de *Design Science* hay que entender los componentes que lo forman: el objeto de estudio y sus dos principales actividades. Por un lado, el objeto de estudio se define como un artefacto en un contexto; por otro, sus dos principales actividades, que son diseñar e investigar el artefacto en su contexto. Podemos, por lo tanto, definir *Design Science* como el diseño e investigación de artefactos en su contexto.

Nótese que un artefacto por sí mismo no resuelve un problema, sino que es la interacción de dicho artefacto con el entorno lo que contribuye a resolver el problema. De ahí que sea tan importante entender el dominio ya que, dependiendo de este, un artefacto puede contribuir a solucionar un problema o no.

Dicha metodología contribuye a la base de conocimiento actual mediante la creación de artefactos que solucionan problemas previamente analizados. Estos problemas, pueden ser problemas de diseño (*Design Problems*) o cuestiones de conocimiento (*Knowledge Questions*).

El contexto del problema puede ser expandido con los *stakeholders* (o actores, todas aquellas personas que estén interesadas en el proyecto o participen en él)

del artefacto y el conocimiento utilizado para diseñar dicho artefacto. Este contexto ampliado constituye el framework de *Design Science*. En él se identifican tres objetos:

- Contexto social: contiene los stakeholders relacionados con el proyecto así como los sponsors. De este contexto se obtienen los objetivos del proyecto.
- *Design Science*: la propia metodología *Design Science*.
- Contexto de conocimiento: formado por el conocimiento ya existente, puede incluir teorías existentes, especificaciones de diseños ya realizados, lecciones aprendidas, etc. Los proyectos de *Design Science* se nutren de este conocimiento (conocimiento previo) y pueden contribuir al él mediante la generación de nuevos diseños o la respuesta a preguntas de conocimiento (conocimiento posterior).

Los proyectos de *Design Science* se inician identificando las necesidades de los stakeholders que dan a lugar a las preguntas de investigación. Estas preguntas son resueltas y contribuyen al conocimiento del contexto.

Los problemas, ya sean de ingeniería (se define como la diferencia entre la forma en la que se percibe el problema por parte de los actores y cómo les gustaría que fuese realmente) o conocimiento (se define como la diferencia entre el conocimiento actual de los actores sobre el mundo y lo que les gustaría conocer), se descomponen en subproblemas de ingeniería y conocimiento que quedan anidados dentro del problema principal.

Design Problems Implica un cambio en el mundo real y requiere un análisis de los objetivos de los stakeholders. No hay una única solución posible y estas deben evaluarse conforme a los objetivos definidos de los stakeholders. Su objetivo es mejorar un problema existente en el contexto mediante la generación de un artefacto que satisfaga algunos requerimientos con el fin de ayudar a los stakeholders a alcanzar sus objetivos. Se resuelven en los ciclos de diseño.

Knowledge Questions No intentan realizar un cambio, sino que realizan cuestiones de conocimiento sobre el mundo real tal y como es, es decir, realiza preguntas sobre el contexto sin proponer una mejora. Al contrario que con los problemas de diseño, la respuesta (una hipótesis) es una y solo una. Se resuelven en los ciclos empíricos. Existen dos tipos de preguntas de conocimiento: empíricas (requieren datos sobre el mundo para ser contestadas) y analíticas (se responden mediante análisis conceptual).

Un proyecto de *Design Science* está formado por círculos regulativos que pueden ser de dos tipos:

- Ciclo de diseño (*Design Cycle*): un ciclo de diseño itera sobre actividades de diseño e investigación. Se trata de un proceso que consiste en las siguientes tareas:

  - Investigación del problema: se deben identificar los stakeholders, sus objetivos y entender el dominio. ¿Qué se desea mejorar? ¿Cómo va a mejorarse?
  - Diseño de la solución: en esta fase se realiza la especificación de requisitos y cómo contribuyen a los objetivos, se analizan soluciones existentes y se diseñan nuevas soluciones.
  - Validación de la solución: ¿produce los efectos deseados la solución propuesta? ¿satisface los requerimientos? El objetivo es validar si la solución contribuye a la consecución de los objetivos de los stakeholders.

- Ciclo empírico (*Empirical Cycle*): es el ciclo utilizado para responder a preguntas de conocimiento. Este ciclo se divide en las siguientes tareas:

  - Investigación del problema: determina cuál es el problema a resolver.
  - Diseño de la investigación: ¿Qué actividades van a ser realizadas para solucionar dicho problema de conocimiento?
  - Validación de la investigación: se analiza la validez tanto de la investigación como de los métodos definidos para realizarla.
  - Ejecución de la investigación: se realiza la investigación según el plan definido anteriormente.
  - Análisis de los resultados: se analizan los datos obtenidos tras la realización de la investigación

Para resolver un problema es necesario determinar de qué tipo de problema se trata; si encontramos un problema de diseño realizamos un ciclo de diseño. En primer lugar, se investiga el problema a resolver (*Problem Investigation*); se debe describir el contexto de dicho problema, así como las posibles mejoras aplicables a dicho problema con el objetivo de mejorar las necesidades de los stakeholders y cumplir así sus objetivos.

Una vez caracterizado el problema, se procede al diseño de la solución (*Treatment Design*). Se define como la introducción de un artefacto, que puede ser de cualquier tipo (en nuestro caso se trata de un artefacto software: una aplicación) con el objetivo de ayudar a los stakeholders a cumplir sus objetivos. Durante este proceso se debe investigar el dominio para verificar que no existe una herramienta que resuelva el problema identificado; en caso de que exista, se deben definir posibles mejoras a las herramientas existentes.

Finalmente, es necesario validar la solución (*Treatment Valdiation*) para así proporcionar evidencia de que efectivamente cumple los objetivos de los stakeholders o los ayuda a acercarse a ellos.

Si, por el contrario, se trata de una pregunta de conocimiento, se inicia un ciclo empírico con el objetivo de caracterizar en detalle el problema a investigar. Una vez estudiado el problema, se procede a diseñar cómo se va a llevar a cabo la investigación. Esto se hace especificando cómo se van a recoger los datos, el entorno, los instrumentos y los métodos de análisis de los datos obtenidos tras realizar la investigación.

El siguiente paso, tras el diseño de la investigación, se debe validar dicho diseño para identificar amenazas a la validez del diseño de la investigación. Una vez validada la investigación, se procede a realizar la investigación en la fase de ejecución de la investigación. Por último, se estudian los resultados obtenidos de la investigación y se extraen las conclusiones pertinentes.

2.2 Genética y Genómica

El entendimiento del genoma humano es uno de los grandes desafíos de nuestros tiempos. La complejidad de este dominio es inmensa y no hemos sino empezado a vislumbrar el conocimiento que podemos obtener de él. Para entender el presente es necesario conocer el pasado, es por ello que resulta de especial interés conocer cómo y cuándo se empezaron a realizar los primeros pasos de la genética y la genómica.

Los orígenes de la genética se remontan al siglo XIX. En 1866 el austríaco Gregor Mendel publica las leyes de la herencia [3]. Posteriormente, Albrecht Kossel descubre los ácidos nucleicos en el año 1871. Con la entrada del siglo XX se realiza la primera secuenciación de una proteína, la insulina (1951) y de ADN (1977). Posteriormente, en 1986 se consigue la automatización de la secuenciación del ADN. A finales de siglo se lanza el Proyecto del Genoma Humano (PHG) con el objetivo de secuenciar por completo el genoma humano; este hito se conseguiría en el año 2003.

Con el paso de los años, las tecnologías se han ido perfeccionando y ha permitido la secuenciación en masa no sólo de humanos, sino de cualquier organismo a un precio cada vez más reducido hasta llegar a precios impensables hace una década. No obstante, en ciertas ocasiones, y por limitaciones técnicas (entre otras) se pueden utilizar *scaffolds* a la hora de realizar secuenciaciones. Se trata de secuencias ordenadas, pero no contiguas, que se encuentran separadas por huecos de longitud variable y conocida. [4].

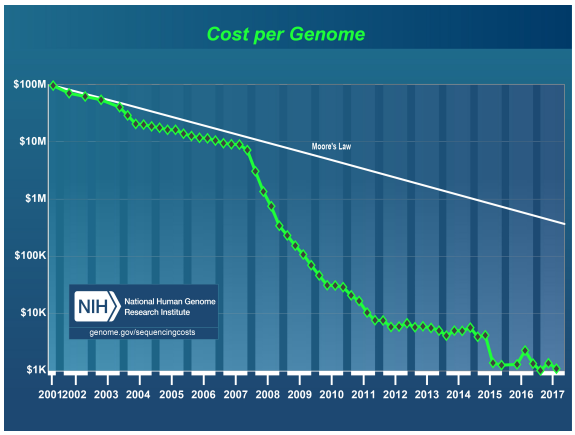

Figura 2.1: Evolución del coste de secuenciación del genoma [5]

En esta breve introducción han sido mencionados múltiples términos que merecen especial atención (genoma, proteína, ADN, etc.). Partamos desde el principio. Todos estos términos vienen referidos en el ámbito de la genética y la genó-

mica, pero, ¿cómo se definen dichos términos? Apoyándonos en el conocimiento actual [6]:

- **Genética** se refiere a un término que define el estudio de los genes y su rol en la herencia: cómo ciertos atributos o condiciones son heredados de una generación a otra. La genética engloba el estudio científico de los genes y sus efectos. Estos genes contienen las instrucciones para fabricar proteínas las cuales afectan de manera directa a las células y las funciones que estas desempeñan en nuestro cuerpo.

- **Genómica** es un término más reciente y define el estudio del genoma completo, incluyendo las interacciones de los genes entre sí y con su entorno. Un ejemplo sería el estudio de enfermedades complejas como enfermedades cardíacas, diabetes o asma, ya que estas enfermedades están causadas por la interacción de diversos genes entre ellos y con factores ambientales.

Son dos los términos clave de estas definiciones: gen y genoma. El genoma se define como el conjunto de instrucciones genéticas presentes en una célula. En el caso del genoma humano, está formado por veintitrés pares de cromosomas, así como un cromosoma presente en las mitocondrias de las células. En total, se estima que los veintitrés pares de cromosomas contienen tres mil millones de bases [7].

Un cromosoma es un conjunto ordenado de ADN presente en el núcleo de la célula, está formado por la unión de dos cromátidas unidas por el centrómero. Actúa como estructura para almacenar el ADN. Forman una estructura sofisticada que contiene los elementos necesarios para realizar procesos de replicación y segregación [8].

El gen es la unidad fisiológica básica de la herencia, contienen la información relacionada con rasgos específicos. Los genes se encuentran organizados, consecutivamente, en cromosomas. Se estima que en humanos existen aproximadamente veinte mil genes. La longitud de los genes es muy variable, yendo desde tan sólo unos cientos de bases a varios miles. Normalmente, contienen información para codificar una proteína [9].

Un gen está formado por dos elementos: intrones y exones. Un intrón es una porción de un gen que no se codifica a aminoácidos [10]. Por otro lado, un exón es una porción de un gen que sí codifica aminoácidos [11]. A su vez; como se puede apreciar en la Figura 2.2, un exón se puede dividir en los siguientes elementos:

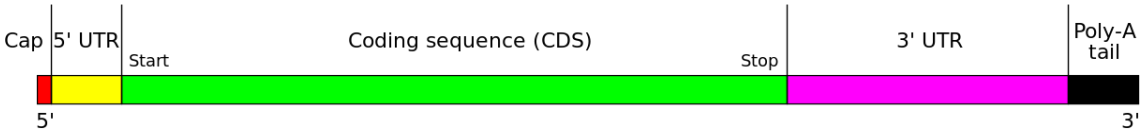

Figura 2.2: Estructura de un exón

- **región no traducida cinco prima (5′ cap)**: nucleótidos situados al final de la región cinco prima. Tiene vital importancia en el proceso de creación de ARN mensajero estable y maduro.

- **caperuza cinco prima (UTR 5')**: es la región que se encuentra antes de la región codificante. En ciertas ocasiones, esta región puede ser traducida parcial o totalmente junto con la región codificante.

- **región codificante (CDS)**: es la región que codifica una proteína.

- **región no traducida tres prima (UTR 3')**: Es la región que se encuentra después de la región codificante, inmediatamente después del codón de terminación. Es frecuente encontrar regiones regulatorias que influyen la expresión génica en esta región.

- **Poliadenilación (Poly-A)**: adición de múltiples bases (Adenina). Es importante para para la exportación nuclear y mejora la estabilidad del ARN mensajero. Esta sección se degrada con el tiempo hasta que el ARN mensajero se degrada enzimáticamente.

Cromosomas, genes, exones... todas estas estructuras están constituidas en su nivel más básico de ADN pero ¿qué es el ADN? Se trata del ácido nucleico que contiene la información genética. Una molécula de ADN (nucleótidos) está formada por tres elementos [12]:

- **Ácido fosfórico**: cada nucleótido puede tener de uno a tres grupos de ácido fosfórico, su fórmula química es H_3PO_4.

- **Desoxirribosa**: monosacárido de 5 átomos de carbono (una pentosa) derivado de la ribosa. Su fórmula es $C_5H_{10}O_4$.

- **Base nitrogenada**: una de las cuatro bases nitrogenadas que se encuentran en el ADN: Adenina (A), Citosina (C), Guanina (G) y Timina (T)

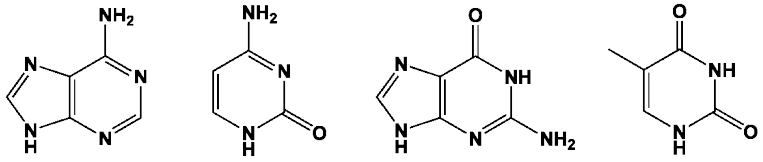

Figura 2.3: Bases nitrogenadas del ADN

Son estructuras de doble hebra y su método de propagación es la autoreplicación. Por otra parte, tenemos las moléculas de ARN, que son de hebra simple y siguen la misma estructura que el ADN aunque, en lugar de Desoxirribosa, emplean la Ribosa como monosacárido (menos estable). En cuanto a sus bases nitrogenadas, ve cambiada la T por Uracilo (U) y se sintetiza a partir del ADN cuando se necesita [13].

Existen diversos tipos de ARN: implicado en la síntesis de proteínas, reguladores o con actividad catalítica. Los detalles de estos exceden el alcance de este TFM. No obstante, cabe resaltar el ARN mensajero (mRNA) [14], que contiene la secuencia de bases que leen los ribosomas [15].

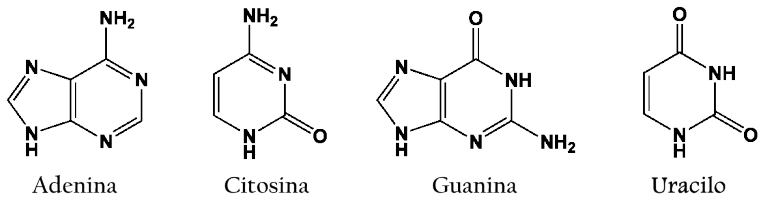

Figura 2.4: Bases nitrogenadas del ARN

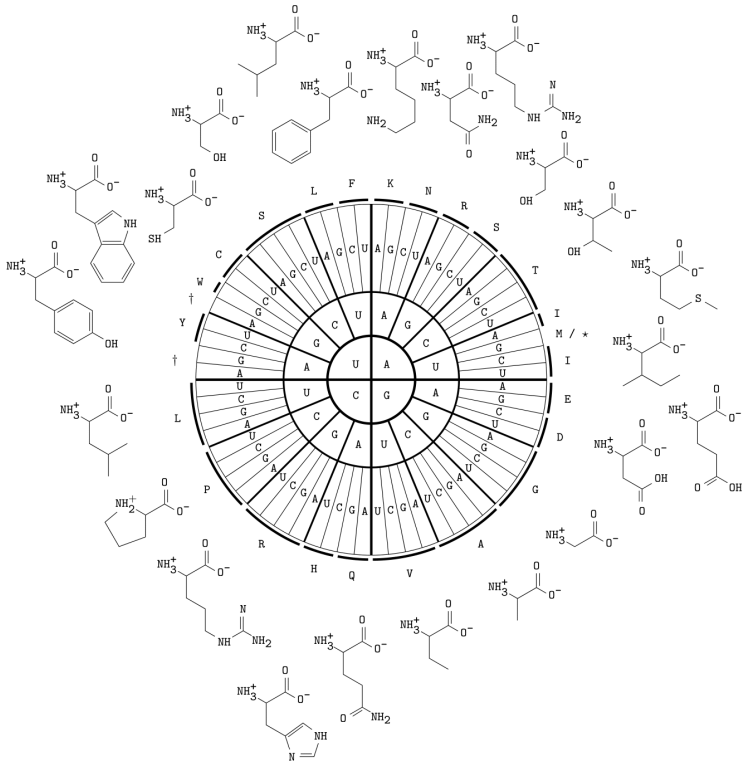

Figura 2.5: Transcripción de codones a aminoácidos

En el proceso de síntesis de proteínas, los ribosomas leen el mRNA para sintetizar proteínas. Estas están formadas por aminoácidos. Cada aminoácido se forma mediante la transcripción de codones [16]. En la Figura 2.5 puede verse una equivalencia entre los codones y los aminoácidos.

Hasta este momento, se ha explicado cómo los genes son secuencias de ADN que, entre otras cosas, contienen la información necesaria para codificar proteínas. El proceso de síntesis parte con la obtención de una cadena de ARN a partir de ADN (exones) para, posteriormente, fabricar aminoácidos recorriendo dicha cadena y leyéndolos en grupos de tres. Una proteína es una molécula formada por un conjunto de aminoácidos dispuestos de una manera concreta [17]. Existen miles de proteínas con funciones muy distintas:

- **Enzimática**: este tipo de proteínas tienen una función catalizadora (incrementan la velocidad) en reacciones químicas. Las enzimas realizan la mayoría de las reacciones metabólicas y son altamente específicas, afectando muy pocas reacciones químicas cada una. Por ejemplo la lactasa o la lipasa.

- **Comunicación celular**: son proteínas relacionadas con la transmisión de señales entre tejidos, también actúan como receptores de señales moleculares y biológicas. Un ejemplo de este tipo de proteína es la insulina.

- **Estructural**: proporciona fuerza y rigidez a distintos componentes y tejidos biológicos. Por ejemplo, el colágeno. También pueden actuar como proteínas motoras, que son capaces de generar fuerza mecánica como la kinesina.

Las enzimas actúan como catalizadores en los *pathways* biológicos. Un pathway se define como un conjunto de acciones producidas en la célula que produce un nuevo producto o realiza un cambio en esta [18]. Pueden desencadenar la formación de nuevas moléculas, como grasas o proteínas o activar o desactivar genes entre otras funciones. Los *pathways* controlan cómo reaccionamos ante el mundo, pues nuestras células están recibiendo constantemente señales químicas tanto desde nuestro cuerpo como del exterior: lesiones, infecciones, estrés o déficit alimentario son algunos ejemplos. Para ajustarse a dichas señales y reaccionar correctamente las células envían y reciben señales a través de los *pathways*. Son responsables de mantener el equilibrio al caminar, controlar cómo y cuándo se abre y cierra la pupila, etc. Pueden actuar tanto en distancias cortas como largas. Cuando un *pathway* no funciona correctamente puede dar como resultado enfermedades tales como el cáncer o la diabetes. Existen muchos tipos de *pathways* pero las que mejor se conocen son:

- **Metabólicos**: hacen posible las reacciones químicas en nuestro cuerpo, por ejemplo, en el proceso de degradación de alimentos para convertirlos en moléculas.

- **Regulatorios de los genes**: activan y desactivan genes. Es de vital importancia ya que los genes contienen la información para la creación de proteínas.

- **Transmisores de señales**: transmiten una señal del exterior de las células a su interior a través de los receptores que estas poseen.

A lo largo de todo nuestro genoma pueden aparecer cambios en las bases que lo conforman. A estos cambios se les llama mutaciones [19]; una mutación es un cambio en la secuencia del ADN. Las mutaciones pueden ser el resultado de errores en la copia del ADN durante la división celular, la exposición a radiaciones ionizantes o a sustancias químicas denominadas mutágenos, o infección por virus. Las mutaciones de la línea germinal se producen en los óvulos y el esperma y puede transmitirse a la descendencia, mientras que las mutaciones somáticas se producen en las células del cuerpo y no se pasan a los hijos. Se trata simplemente de un error cometido al copiar una secuencia de ADN. Algunas de ellas forman parte del ruido de fondo, ya que el proceso de replicación del ADN no es perfecto, de lo cual debemos estar contentos o no existiría la evolución. Pero una mutación también puede ser inducida por causas como la radiación o por sustancias cancerígenas, de forma que puede aumentar el riesgo de padecer cáncer o defectos congénitos. En el fondo es bastante simple, no es más que una falta de ortografía inducida de la secuencia de ADN.

Existen diversos tipos de mutaciones:

Polimorfismo

Un polimorfismo implica una de dos o más variantes de una secuencia particular de ADN. El tipo más común de polimorfismo implica la variación en un solo par de bases. Los poliformismos también pueden ser de mucho mayor tamaño implicando largos tramos de ADN. Un polimorfismo tiene que ocurrir en al menos una de cada 100 personas. Los poliformismos pueden ser cambios de una sola letra, como una C en vez de T. Pero también podrían ser algo más complejo, como un tramo entero del ADN, el cual está presente o ausente. Éstas también pueden llamarse alteraciones en el número de copia; pero todos son poliformismos. Básicamente, es un término general para hablar de la diversidad en los genomas de una especie [20]. Los Polimorfismos de Nucleótido Sencillo (en inglés, *Single Nucleotide Polimorphisms -"SNPs"-*) son un tipo de polimorfismo que producen una variación en un solo par de bases [21].

Mutación sin sentido

Una mutación sin sentido es la sustitución de un solo par de bases que da lugar a la aparición de un codón de terminación donde previamente había un codón que codificaba para un aminoácido. La presencia de este codón de terminación prematura genera una proteína más corta y probablemente no funcional [22].

Mutación puntual

Una mutación puntual se produce cuando se altera un solo par de bases. Las mutaciones puntuales pueden tener uno de los tres efectos siguientes. En primer lugar, la sustitución de una base puede ser una mutación silenciosa, es decir, el codón alterado produce el mismo aminoácido. En segundo lugar, la sustitución de base puede ser una mutación sin sentido en que el codón alterado da lugar a un aminoácido diferente. En tercer lugar, la sustitución de una base puede producir una mutación sin sentido y el codón alterado puede corresponder a una señal de terminación [23].

Mutación con cambio de sentido

Una mutación en el cambio de sentido de lectura se produce cuando el cambio de un solo par de bases da lugar a la sustitución de un aminoácido en la proteína resultante. Esta sustitución de un aminoácido puede no tener ningún efecto, o puede dar lugar a una proteína no funcional [24].

Mutación con cambio del marco de lectura

Una mutación en el cambio del marco de lectura es un tipo de mutación que implica la inserción o deleción de un nucleótido en el que el número de pares de bases eliminado no es divisible por tres. "Divisible por tres" es importante porque, como hemos visto anteriormente, la célula lee un gen en grupos de tres bases. Cada grupo de tres bases corresponde a uno de los 20 aminoácidos diferentes usados para construir una proteína. Si una mutación interrumpe este marco de lectura, entonces toda la secuencia de ADN siguiente a la mutación se lee incorrectamente [25].

2.3 Caso de Estudio

El IVIA es una entidad autónoma de la Generalitat con personalidad jurídica propia. Su misión es la de impulsar la investigación científica y el desarrollo tecnológico en el sector agroalimentario valenciano e integrar esta contribución al progreso de la ciencia agraria en el sistema de relaciones de colaboración y cooperación propiode la actividad investigadora. Entre sus funciones generales destacan:

- Promover y realizar programas de investigación propios o concertados, relacionados con el sector agroalimentario valenciano.
- Transferir los resultados científicos obtenidos y mantener relaciones con el sector agroalimentario para conocer sus necesidades de I+D+I.
- Fomentar las relaciones con otras instituciones de la comunidad científica, tanto nacionales como extranjeras.
- Promover la organización de congresos en temas de interés para la Comunidad Valenciana relacionados con el sector agroalimentario.
- Asesorar en materia de investigación y desarrollo agroalimentario a los órganos dependientes de la Generalitat Valenciana y a las empresas del sector agroalimentario que lo soliciten.
- Contribuir a la formación de personal investigador en el ámbito de sus fines científicos.

Sus objetivos se centran en el desarrollo de programas de mejora vegetal y ganadera que permitan una mayor resiliencia y adaptación de la producción agraria valenciana aumentando su diversificación y competitividad. Estos programas buscan prevenir las plagas y enfermedades potenciales y emergentes y controlar las que afectan actualmente a la agricultura valenciana reduciendo el impacto ambiental de los métodos de control; contribuir a garantizar la sostenibilidad y diversidad de las producciones agrarias y ganaderas valencianas; mejorar los sistemas de control de enfermedades y la calidad de poscosecha de los productos hortofrutículas para consumo en fresco y fomentar las fórmulas de colaboración y las sinergias con el sistema científico y tecnológico nacional e internacional.

Como se observa, el IVIA tiene numerosas líneas de investigación en ámbitos muy variados. Nos centraremos en la línea de trabajo con la que se ha colaborado: **Obtención, mejora y conservación de material vegetal** del centro de cítricos y producción vegetal y centro de genómica. Dentro de esta línea de trabajo se encuentran las investigaciones centradas en nuevas variedades de cítricos (clementinas) por irradiación y selección dirigida por métodos genómicos, así como la caracterización genotípica y agronómica de variedades, realizando estudios de asociación genotipo-fenotipo.

Trabajan con una gran variedad de fuentes de datos con formatos muy dispersos que siguen estándares diferentes. Se observa un problema en dos dimensiones: la primera de ellas, como hemos nombrado ya, referente a la variabilidad de

los datos. No hay un único repositorio de datos del que nutrirse y además, a estos ficheros obtenidos de fuentes externas se añaden datos de producción propia. La segunda dimensión consiste en el volumen de estos datos. Por cada variedad estudiada se generan gigas de información. Esta información es difícil de procesar debido a su complejidad y a todas las variables que se han de tener en cuenta. Si, además, se tiene en cuenta el número de variedades con las que se trabaja (unas ochenta aproximadamente), se observa un volumen de datos no analizable de manera manual o sin soporte de una herramienta adecuada.

Los estudios de asociación genotipo-fenotipo son el principal objeto de mejora. Existen dos posibles aproximaciones. En la primera de ellas, se seleccionan grupos de cítricos y se comparan para obtener en qué se diferencian a nivel genético (variaciones en su genoma). Una vez se sabe qué las diferencia a este nivel se procede a buscar qué efectos tienen en su fenotipo. Por ejemplo, analizar un conjunto de variaciones presentes en un grupo de variedades que afectan a un gen y analizar qué rutas metabólicas (*pathways*) se ven alteradas al, por ejemplo, modificar una proteína que interviene en dicho *pathway*. La segunda es partir de alguna anotación funcional de un gen (pueden ser enzimas, proteínas, grupos ortólogos[1], etc.) e identificar las variaciones concretas que se relacionan con alguna de estas anotaciones funcionales. Como se puede ver, la primera opción parte del genotipo y genera conocimiento referido al fenotipo y la segunda opción sigue el camino contrario y partiendo del fenotipo se genera conocimiento del genotipo.

2.4 Participantes y Objetivos

En esta sección vamos a definir los diferentes stakeholders involucrados en este proyecto. Los stakeholders son una fuente de requisitos, restricciones y requerimientos para el proyecto y es por ello que resulta de gran importancia identificarlos correctamente.

El objetivo final de la fase de diseño de la solución en el ciclo de diseño es generar una mejora para los stakeholders cuando dicha solución sea aplicada en el contexto. No obstante, también es posible que existan conflictos entre los diferentes stakeholders en cuyo caso se hace necesario llegar a un punto intermedio que satisfaga a todas las partes en la medida de lo posible.

Se han identificado los siguientes stakeholders:

- IVIA: usuario final y principal fuente de objetivos, restricciones y requerimientos.
- Centro de Investigación en Métodos de Producción de Software (PROS): centro de investigación en el cuál se va a realizar el proyecto.
- Autor de la TFM: como desarrollador, es el encargado de diseñar y validar la solución.

[1]Se trata de grupos de genes pertenecientes a distintas especies que han evolucionado a partir de un ancestro común y que suelen mantener la misma función a través del tiempo. Para más información consultar Glosario

– Tutores: realizan el rol de consultores durante el proyecto.

Es importante distinguir entre los objetivos de los stakeholders (detallados a continuación), los objetivos del investigador (Sección 1.2) y los objetivos del proyecto *Design Science* (Sección 2.5) ya que estos pueden diferir

Nos vamos a centrar en el IVIA para identificar correctamente sus objetivos ya que a partir de estos se generarán las preguntas de investigación. IVIA es un centro de investigación y como tal está interesada en la producción científica y realizar publicaciones. Para ello, dispone de un grupo que se dedica al estudio genómico de las distintas variedades de cítricos (más detalles sobre sus características y limitaciones en la Sección 2.3). Debido a la naturaleza de los datos que utilizan y a las limitaciones técnicas de los medios con los que trabajan, en múltiples ocasiones no les es posible realizar los estudios necesarios para generar nuevos resultados. A raíz de esta situación, identifican una clara necesidad de mejora:

"Mejorar las herramientas de análisis de datos genómicos actuales para aumentar su eficiencia y rapidez"

2.5 Preguntas de Investigación

En esta sección se formulan las preguntas de investigación respondidas en este TFM.

Como hemos visto en la Sección 2.4, el principal stakeholder tiene un único objetivo muy claro. Partiendo de este objetivo hemos obtenido los objetivos de la investigación y, a partir de estos, las preguntas de investigación.

Dado el objetivo del stakeholder, el primer paso es recabar información del dominio de interés (genómico-cítrico). En primer lugar, se deben estudiar los datos de dicho dominio, su estructura, sus características, etc. para entender las características de los datos y enlazarlos con las limitaciones del IVIA. Podemos definir este objetivo como un objetivo de conocimiento (*Knowledge goal*). De este objetivo aparece la primera pregunta de investigación (RQ_1)

Una vez establecido y estudiado el dominio del problema, proponemos la creación de un esquema conceptual del genoma de los cítricos con el objetivo es establecer las distintas entidades existentes y sus relaciones entre ellas. Este esquema ayudará a mejorar el entendimiento del dominio y proporcionará una base sólida sobre la que seguir trabajando. Al generarse un nuevo artefacto, obtenemos un objetivo de diseño de artefacto que genera la segunda pregunta de investigación (RQ_2)

Partiendo del esquema generado como base de conocimiento, proponemos la construcción mediante desarrollo dirigido por modelos (MDD)[26] de una aplicación que permita satisfacer las necesidades existentes entre los expertos en el dominio en la realización de análisis de datos en el ámbito genómico-cítrico. Se trata de un objetivo de diseño de artefacto que nos conduce a la tercera pregunta de investigación (RQ_3). Se deben estudiar los beneficios de aplicar técnicas de MDD y la arquitectura óptima para aplicar a la solución de modo que se asegure

una mejora de las limitaciones en el contexto estudiado. Factores como la productividad o la satisfacción de uso de la solución por parte de los usuarios finales son tenidos en cuenta.

RQ1 [KQ] — ¿Qué características definen el tipo de datos involucrados en el ámbito genómico-cítrico y cómo trabajan con ellos en el IVIA?
RQ2 [DP] — Diseño de un esquema conceptual del genoma de los cítricos
RQ3 [DP] — Diseño de una aplicación que mejore el proceso de análisis genético por el IVIA en el ámbito genómico-cítrico

Tabla 2.1: Preguntas de investigación

Tras determinar las preguntas de investigación estas serán resueltas a lo largo del documento. La primera pregunta de investigación será tratada en las Subsecciones 3.1 y 4.2. Donde se caracterizarán y detallarán los datos y se explicará cómo trabajan con ellos en el IVIA respectivamente. La segunda pregunta de investigación se trata en la Subsección 3.1, donde se detallará todo el proceso de diseño del esquema conceptual. Se obtiene como artefacto resultante el ECGCit. que servirá de ayuda en todo el proceso de diseño e implementación de la herramienta de análisis genómicos. Finalmente, la tercera y última pregunta de investigación se trata a lo largo de toda la Sección 3 donde se realiza el diseño e implementación del artefacto resultante de dicha pregunta de investigación.

Parte II

Desarrollo

CAPÍTULO 3
Diseño de la Solución

En este capítulo se presenta el diseño e implementación de la solución adoptada. Esta solución está basada en la producción de dos artefactos: un esquema conceptual de la información genómica de los cítricos (ECGCit.) y una aplicación web de análisis de datos genómicos respaldado por el apoyo ontológico del esquema generado. Se divide en cinco secciones: la primera (Sección 3.1) expone el proceso seguido en la creación del ECGCit. A continuación, (Sección 3.2) se especifican los requisitos de la aplicación web a desarrollar. La tercera sección (Sección 3.3) presenta el proceso de selección del SGBD y, una vez este ha sido seleccionado, el proceso ETL realizado para la creación de la base de datos. En la cuarta sección (Sección 3.4) se especifica el API que ofrece la aplicación y se detalla la arquitectura del servidor. Por último, en la quinta sección, (Sección 3.5) se muestra la interfaz de usuario generada para interaccionar con el servidor y realizar los análisis.

3.1 Esquema Conceptual del Genoma de los Cítricos

En este capítulo se presenta el esquema conceptual diseñado y el proceso seguido en la generación del mismo. Para el diseño de dicho esquema se han tomado los siguientes elementos como fuente de entrada de información:

- Los ficheros de datos proporcionados por el IVIA.
- Fuente de origen de los ficheros obtenidos.
- Información extraída durante las reuniones realizadas con el IVIA.
- Conocimiento propio.

Los ficheros proporcionados pueden clasificarse según su tipo. Existen cuatro bloques basándonos en este criterio:

- **FASTA**: se trata de ficheros con extensión “fa”. Son ficheros de texto utilizados para representar secuencias de ácidos nucleicos. Además, es posible incluir nombres de secuencias y comentarios. Se trata de un formato simple

que ofrece una mayor comodidad a la hora de su manipulación y análisis. Consta de una línea de cabecera donde se aloja el identificador, el nombre y los comentarios y en la siguiente línea comienza la secuencia biológica obtenido a partir de la herramienta software FASTA [27].

- **Genetic Feature Format Version 3 (GFF3)**: se trata de ficheros con extensión "gff3". Son ficheros de texto plano que constan de nueve columnas las cuales están delimitadas por tabulaciones. Los detalles de las columnas que componen estos ficheros se verán en detalle a continuación. [28].
- **Texto plano**: además de los ficheros ya nombrados, se han utilizado varios ficheros de extensión "txt" que no siguen ningún estándar definido, sino que se han descargado de fuentes de datos externas como ficheros tabulares.
- **VCF**: ficheros con extensión "vcf". Son ficheros de texto que almacenan variaciones e información sobre las mismas. Contiene un encabezado con metainformación, una cabecera y la información propiamente dicha [29].

Conforme se presenten los diferentes ficheros y sus características, se van a identificar de manera progresiva tanto los elementos como las relaciones que conforman las unidades conceptuales básicas del esquema conceptual que queremos elaborar. Por cada fichero se ha analizado su estructura y origen y se explican los conceptos relevantes junto con sus atributos y relaciones.

Fasta

Estos ficheros de formato simple se obtienen al ejecutar el *software* homónimo y contiene secuencias de ADN y proteínas identificadas. El proceso de alineamiento de secuencias se realiza mediante el algoritmo Smith-Waterman [30]. Existen dos ficheros en formato FASTA; el primero contiene las secuencias, formadas por nucleótidos de los *scaffolds* (conjunto de secuencias de nucleótidos no contiguas cuya separación es de longitud desconocida) identificados. El segundo contiene las secuencias de proteínas.

La utilización de *scaffolds* en lugar de cromosomas es debido a que cuando la técnica de secuenciación produce errores, se obtienen secuencias cuya posición cromosómica es desconocida. Si la técnica no produjera ningún error de lectura, se obtendrían nueve *scaffolds* (uno por cada uno de los cromosomas que contiene la mandarina) pero no es el caso, ya que se han identificado mil trescientos noventa y ocho (cuanto mayor es el identificador del *scaffold*, menor es su longitud). Además, se presentan bases con la letra N que representan posiciones para las cuales el nucleótido es desconocido:

- *Cclementina_v1.0_scaffolds.fa* Contiene información relativa a la secuencia de cada uno de los *scaffold* secuenciados. Las líneas de cabecera empiezan mediante el caracter ">" e identifica el *scaffold*, a partir de la siguiente línea aparece la secuencia completa que forma dicho *scaffold* hasta la siguiente línea de cabecera (Tabla 3.1). De este fichero el único concepto que se extrae es el de *scaffold*, que tiene como únicos atributos su identificador y la cadena de bases que lo forma. No se obtiene ninguna relación (Tabla 3.2).

>{identificador} {secuencia}
>scaffold_1 AAAAATAAGCGCCCGAAGGTCGTGGTGCCCAAACCCCCCAGCGGAAA . . . CGTCCGCCGGACTCGGAATGCGGCGAGACTTTGCGAGGGGCCTCGCTG

Tabla 3.1: Formato del fichero Cclementina_v1.0_scaffolds.fa

Elementos	
Concepto	**Atributos**
scaffold	*identificador* *secuencia*

Tabla 3.2: Datos extraídos de Cclementina_v1.0_scaffolds.fa

– *Cclementina_182_v1.0.protein.fa* Contiene secuencias de proteínas, así como información relativa a la misma. Cada elemento se divide en dos partes (mismo patrón del fichero anterior): una línea de cabecera y la secuencia. En la línea de cabecera se encuentran seis atributos: nombre, pacid, transcript, locus, ID y annot-version. Los atributos nombre, transcript e ID identifican el mRNA que codifica la proteína, el atributo locus identifica el gen en el cual se localiza el mRNA, annot-version indica la versión (en nuestro caso siempre la 1.0), por último, el atributo pacid (*Phytozome Accession Number*) es un numérico que identifica el mRNA utilizado en la base de datos Phytozome[1] (Tabla 3.3).

>{Nombre} pacid={pacid} transcript={transcript} locus={locus} ID={ID} annot-version={annot-version} {Secuencia}
>Ciclev10013963m pacid=20785355 transcript=Ciclev10013963m locus=Ciclev10013963m.g ID=Ciclev10013963m.v1.0 annot-version=v1.0 MTATAFCLTGLLITRWVYSRHFPLSDLYESLIFLSWSFSIIHKIFDFKNNQNHL SAITAPSAFFTQGFATSGFLTKMHQSRILVPALQVQWLMMHVFFREQYGLR RGGLIGIGIQKTPPPFITWTIFGIYLHTRTNTKWEGVNPAIVASMGFFIIWICY FGVNLLGIGFHSYGPFN*

Tabla 3.3: Formato del fichero Cclementina_182_v1.0.protein.fa

[1] https://phytozome.jgi.doe.gov

Por cada entrada en el fichero se identifica una proteína con tres atributos y un gen (contiene los elementos que traducen el mRNA) y mRNA (que codifica dicha proteína). Así pues, se identifican tres relaciones entre los elementos. Por un lado, tanto el gen como el mRNA se relacionan con la proteína; el mRNA porque codifica la proteína y el gen porque contiene lla secuencia que se traduce en el mRNA. Por otro lado, como acabamos de observar, existe una relación entre gen y mRNA. La proteína es codificada únicamente por un mRNA, por lo tanto, se relaciona con un mRNA y un gen. Del mismo modo, un mRNA codifica únicamente una proteína, pero un gen puede codificar más de una proteína (Tabla 3.4).

Elementos	
Concepto	**Atributos**
proteína	*ID* *secuencia* *annot-version*
gen	*ID*
mRNA	*ID* *nombre* *transcript* *pacid*
Relaciones	
Concepto 1	**Concepto 2**
gen	proteína
mRNA	proteína
mRNA	gen

Tabla 3.4: Datos extraídos de Cclementina_182_v1.0.protein.fa

GFF3

De formato gff3 se disponen de dos ficheros que contienen la misma información, con la diferencia de que el segundo de ellos (el que hemos analizado) incluye los exones y el primero no, razón por la cual ha sido descartado al añadir menos información. El fichero ha sido obtenido de la web[2]. Consiste en el genoma completo de la *Citrus Clementina* (mandarina) en su versión 1.0. Está formado por mil trescientos noventa y ocho *scaffolds* y cuenta con un 2.1 % de bases cuyo valor es desconocido y una cobertura de valor 7.0x (número de lecturas únicas para incluir los nucleótidos). Se ha utilizado una planta haploide (que contiene un único conjunto de cromosomas, al contrario que diploide) para simplificar el proceso de secuenciación y ensamblado [31].

[2] https://www.citrusgenomedb.org

– *Cclementina_182_v1.0.gene_exons.gff3* Se trata de un fichero compuesto por nueve columnas; tal y como lo especifica el estándar del formato, las columnas se describen como sigue (Tabla 3.5):

- seqid: identificador del punto de referencia utilizado para localizar el elemento. En otras palabras, el *scaffold* al que pertenece.
- source: campo descriptivo que informa sobre la fuente o algoritmo utilizado para generar el elemento; normalmente se indica la herramienta software utilizada.
- type: indica el tipo del elemento. Debe ser un término presente en *Gene Ontology*[3]
- start: Posición de inicio del elemento.
- end: Posición de fin del elemento utilizando como referencia el valor de la segunda columna. El valor debe ser igual o superior al valor de la columna Inicio.
- score: número decimal que indica el valor de p (concepto estadístico relacionado con el nivel de significación y utilizado para rechazar hipótesis) de la variación (en nuestro caso no se dispone de dicho valor).
- strand: indica el sentido del elemento, que puede ir de 5′ a 3′ (+) o de 3′ a 5′ (-).
- phase: para aquellos elementos cuyo tipo sea región codificante, indica las bases que han de ser eliminadas para alcanzar el primer codón. Su valor oscila entre cero (el codón empieza en la primera base) y dos (el codón empieza en la tercera base).
- attributes: lista de atributos del elemento. Incluye el identificador, nombre, alias, súper elemento al que pertenece el elemento o notas informativas.

{seqid} {source} {type} {start} {end} {score} {strand} {phase} {attributes}
scaffold_1 phytozomev10 gene 23527053 23542559 . + . ID=Ciclev10007219m.g.v1.0;Name=Ciclev10007219m.g

Tabla 3.5: Formato del fichero Cclementina_182_v1.0.gene_exons.gff3

El atributo type tiene seis posibles valores en el fichero: gene, mRNA, exon, five_prime_UTR, CDS y three_prime_UTR. Los atributos varían en función de qué tipo de elemento es; así, para los de tipo gene los atributos que contiene son:

[3]Consorcio cuya misión consiste en el desarrollo de un modelo computacional de sistemas biológicos. Disponible en: http://www.geneontology.org/.

- ID: identificador del elemento. Es un valor obligatorio para aquellos elementos de los cuales cuelgan hijos. Este valor debe ser único en todo el fichero.
- Name: el nombre con el que se muestra el elemento a los usuarios. Al contrario que con el ID, no se requiere que sea un valor único.

Para mRNA además de ID y Name, incluye los siguientes atributos:

- pacid (*Phytozome Accession Number*): identificador numérico del valor del campo Parent (mRNA) utilizado en la base de datos Phytozome[4].
- longest: valor numérico que indica si es el mRNA más largo del gen (1) o no (0).
- Parent: identificador del elemento padre del cual cuelga este elemento. Se utiliza para agrupar elementos, por ejemplo, exones por mRNA. En nuestro caso se utiliza para agrupar los elementos por mRNA

El resto de tipos de elementos (UTR 5', CDS, UTR 3') comparten los atributos: ID, Parent y pacid (carecen de atributo Name y longest).

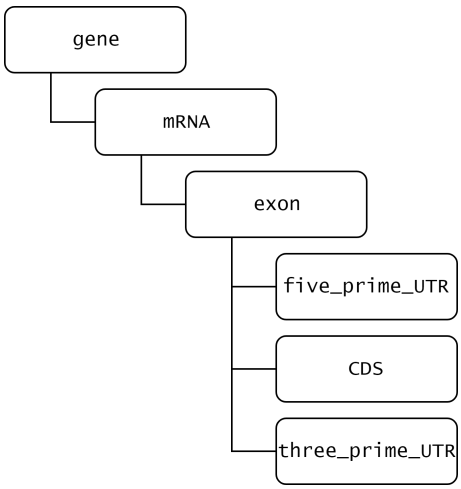

Figura 3.1: Estructura jerárquica de Cclementina_182_v1.0.gene_exons.gff3

En la estructura del fichero se observa un orden recurrente en el cual los elementos guardan un orden jerárquico como se observa en la Figura 3.1. El primer elemento es gene (gen) y hasta llegar al siguiente elemento de tipo gen, todos los elementos están situados dentro del rango de posición de inicio y fin de dicho gen. Lo mismo sucede con los elementos mRNA y exon (a partir del elemento exon, los tres restantes se encuentran al mismo nivel). Es decir, estamos tomando las posiciones para obtener qué elementos forman parte de otros elementos. De este modo se establecen las relaciones nombradas en la Tabla 3.6 (biológicamente hablando un mRNA no está formado por exones, pero si se obtiene mediante la transcripción de estos, por eso mantenemos la relación jerárquica definida).

El campo ID de los elementos también sigue una estructura definida. Todos ellos comparten el mismo inicio de identificador: comienza con la palabra

[4] https://phytozome.jgi.doe.gov

"Ciclev" (Ciclev) seguida de un número de ocho cifras (Ciclev10007219), la letra m y un punto (Ciclev10007219m.). A partir de aquí varía de un elemento a otro. Empezando por el gen, continúa con la letra g seguida de un punto (Ciclev10007219m.g.) y la versión del genoma (Ciclev10007219m.g.v1.0). El mRNA se identifica del mismo modo que el gen salvo que no contiene la letra g (Ciclev10007219m.v1.0). Como un gen puede transcribirse a más de un mRNA, el primer mRNA tendrá en el identificador el mismo número de ocho cifras que el gen mientras que el identificador del resto de mRNAs se irá incrementando (el campo Parent nos indica de qué gen se ha traducido un mRNA). El resto de elementos comparten el identificador del mRNA del que cuelgan, pero se le añade, separado por un punto, el tipo de elemento que es y su orden de aparición. Por ejemplo, para un CDS el identificador quedaría Ciclev10007219m.v1.0.CDS.1.

Por un lado, cada línea del fichero identifica un elemento del tipo que corresponda según el valor de su columna type y se obtienen los atributos identificados durante el proceso de análisis. Todos los elementos poseen los atributos seqid, source, type, start, end, score, strand y phase (identificados como "comunes" en la Tabla 3.6). Por otro lado, y partiendo de la relación de jerarquía explicada anteriormente se procede a la generación de las relaciones. La columna seqid identifica el *scaffold* al cual pertenece cada elemento, por lo que por cada elemento se crea una relación entre dicho elemento y el *scaffold*.

El siguiente nivel es el de gen. Una vez de identifica un gen, hasta llegar al siguiente, todos los elementos se relacionarán con dicho gen. Este proceso se repite para los mRNAs y exones. Por ejemplo, un CDS se relacionará con, como mínimo, un exón, un mRNA, un gen y un *scaffold*.

En cuanto a la cardinalidad de las relaciones, un *scaffold* posee una relación de uno a muchos con el resto de elementos sin excepción. El gen se relacionará exactamente con un *scaffold* y se relaciona una o varias veces con el resto de elementos. El mRNA se relaciona con un *scaffold* y un gen y con uno o más del resto de elementos. El exón se relaciona con uno o varios mRNA debido a la superposición de estos en el genoma y por extensión, con uno o más genes sin embargo se relaciona únicamente con un *scaffold*; además, el exón se relaciona con un CDS y puede o no relacionarse con tanto con un elemento 5' cap como con un elemento UTR 3'. Los tres elementos restantes, 5' cap, CDS y UTR 3' se relacionan con un exón, uno o más mRNAs y exactamente un gen y un *scaffold*.

Elementos

Concepto	Atributos
scaffold	*identificador*
gen	*comunes* *ID* *Name*
mRNA	*comunes* *ID*

Continúa en la página posterior

Continúa de la página anterior

Concepto	Atributos
	Name *pacid* *longest* *Parent*
exón	*comunes* *ID* *Parent* *pacid*
UTR 3′	*véase exon*
CDS	*véase exon*
UTR 5′	*véase exon*

Relaciones

Concepto 1	Concepto 2
scaffold	gen
scaffold	mRNA
scaffold	exón
scaffold	UTR 5′
scaffold	CDS
scaffold	UTR 3′
gen	mRNA
mRNA	exón
exón	UTR 5′
exón	CDS
exón	UTR 3′

Tabla 3.6: Datos extraídos de Cclementina_182_v1.0.gene_exons.gff3

Texto plano

Para estos ficheros es importante tener en cuenta la relación existente entre un gen, un mRNA y una proteína. Dado que el identificador de una proteína es el mismo que el identificador del mRNA que la codifica y este puede ser igual al gen en el cual está presente (si es distinto es sencillo encontrar dicha relación), se puede hablar indistintamente de dichos identificadores en los ficheros expuestos a continuación. Esto se traduce en que, en ciertas ocasiones, ante un identificador dado, no será posible determinar si se trata de un mRNA o una proteína, pero esto nos da igual ya que poseen el mismo identificador. Existen cinco ficheros de texto plano que no siguen ningún estándar definido:

– *Cclementia_v1.0_genes2GO.txt* Se trata de un fichero tabulado que relaciona proteínas (como hemos visto anteriormente, la relación con los genes y los mRNA es directa, de ahí el nombre del fichero) con anotaciones de Gene Ontology (Tabla 3.7) [5].

La primera columna identifica la proteína, la segunda identifica el término de *Gene Ontology* mediante su código, la tercera indica la ontología a la que pertenece el término y por último, la cuarta, proporciona un nombre descriptivo para el término.

{Nombre de la proteína}	{Identificador de Gene Ontology}	{Ontología}	{Nombre}
Ciclev10000001m	GO:0005488	Molecular Function	binding

Tabla 3.7: Formato del fichero Cclementia_v1.0_genes2GO.txt

Se identifica una nueva entidad con la información relativa al término de *Gene Ontology* y se relaciona con la proteína correspondiente. Se trata de una relación de muchos a muchos pues una proteína puede estar relacionada con múltiples términos y cada término puede relacionarse con varias proteínas (Tabla 3.8).

Elementos	
Concepto	**Atributos**
GO	*ID* *ontología* *nombre*
Relaciones	
Concepto 1	**Concepto 2**
proteína	GO

Tabla 3.8: Datos extraídos de ficheros Cclementia_v1.0_genes2GO.txt

– *Cclementia_v1.0_genes2IPR.txt*

Se trata de un fichero tabulado que relaciona proteínas con dominios (secuencia de ADN asociada a una función particular, véase Glosario para más información). La primera columna identifica la proteína, la segunda identifica el dominio y la tercera proporciona un nombre descriptivo para dicho dominio (Tabla 3.9).

[5] Los términos de dicha ontología puede ser encontrada aquí: https://www.ebi.ac.uk/QuickGO/.

{Nombre de la proteína} {Identificador de Dominio} {Nombre del Dominio}
Ciclev10000001m IPR000449 Ubiquitin-associated/translation elongation factor EF1B, N-terminal

Tabla 3.9: Formato del fichero Cclementia_v1.0_genes2IPR.txt

Por cada fila del fichero se identifica un dominio y una relación entre la proteína y este. Un dominio asocia múltiples proteínas y una proteína puede estar, o no, asociada con más de un dominio (Tabla 3.10).

Elementos	
Concepto	**Atributos**
dominio	*ID* *nombre*
Relaciones	
Concepto 1	**Concepto 2**
proteína	dominio

Tabla 3.10: Datos extraídos de ficheros Cclementia_v1.0_genes2IPR.txt

– *Cclementina_v1.0_KEGG-orthologs.txt* Se trata de un fichero tabulado que relaciona genes con grupos ortólogos (Conjunto de genes con un ancestro común, véase Glosario para más información). El acrónimo KEGG proviene de *Kyoto Encyclopedia of Genes and Genomes*[6]. Se trata de una avanzada base de datos con recursos orientados al entendimiento de funciones biológicas a alto nivel. El fichero está formado por tres columnas. El valor de la primera columna se utiliza para localizar al gen. La segunda columna proporciona un identificador para el grupo ortólogo (el identificador inicia con la letra K seguida de cinco números). Finalmente, en la tercera columna se encuentra la descripción del grupo ortólogo, que no es más que un nombre descriptivo el cuál incluye qué enzimas se ven afectadas o relacionadas con los genes pertenecientes al grupo ortólogo (Tabla 3.11).

{Gen} {identificador} {Definición}
Ciclev10000939m K00844 hexokinase [EC:2.7.1.1]

Tabla 3.11: Formato del fichero Cclementina_v1.0_KEGG-orthologs.txt

[6] http://www.genome.jp/kegg/

Por cada línea se identifica el grupo ortólogo y se obtiene una relación entre dicho grupo ortólogo y los genes que forman parte de el. Además, en la descripción de cada grupo ortólogo se indica con qué enzima o enzimas está relacionado. Estas relaciones también se identifican y definen. Tanto los genes como las enzimas pueden relacionarse con más de un grupo ortólogo (Tabla 3.12).

Elementos	
Concepto	**Atributos**
grupo ortólogo	*ID* *descripción*
Relaciones	
Concepto 1	**Concepto 2**
gen	grupo ortólogo
enzima	grupo ortólogo

Tabla 3.12: Datos extraídos de ficheros Cclementina_v1.0_KEGG-orthologs.txt

– Cclementina_v1.0_KEGG-pathways.txt Se trata de un fichero tabulado que relaciona mRNAs con pathways (Tabla 3.13). En la primera columna está el identificador del mRNA; la segunda y tercera columna están relacionados con el pathway ya que muestran su identificador (segunda columna) y la definición del mismo (tercera columna).

{mrna}	{Identificador}	{Definición}
Ciclev10001298m	ko00010	Glycolysis / Gluconeogenesis

Tabla 3.13: Formato del fichero Cclementina_v1.0_KEGG-pathways.txt

El análisis de este fichero nos permite, por un lado, identificar los pathways existentes, y por otro, relacionar elementos de mRNA (y por extensión los genes) con pathways (relación de muchos a muchos) (Tabla 3.14).

Elementos	
Concepto	**Atributos**
pathway	*ID* *descripción*
Relaciones	
Concepto 1	**Concepto 2**
pathway	mRNA

Tabla 3.14: Datos extraídos de ficheros Cclementina_v1.0_KEGG-pathways.txt

– kegg_txt_20110915_1636.txt Fichero tabulado que relaciona enzimas con pathways. Se trata de un fichero más complejo que consta de ocho columnas pudiendo agruparse en dos grupos: un primer grupo con información referente al pathway y un segundo grupo con información referente a la enzima (Tabla 3.15). Al primer grupo pertenece:

  – primera columna (pathway): contiene el nombre descriptivo del pathway.
  – segunda columna (Seqs in Pathway): número de secuencias en el pathway.
  – séptima columna (pathwayId): identificador del pathway.
  – octava columna (PathwayImage): ruta donde se almacena la imagen del proceso del pathway (puede verse un ejemplo de dichas imágenes en la Figura 3.2).

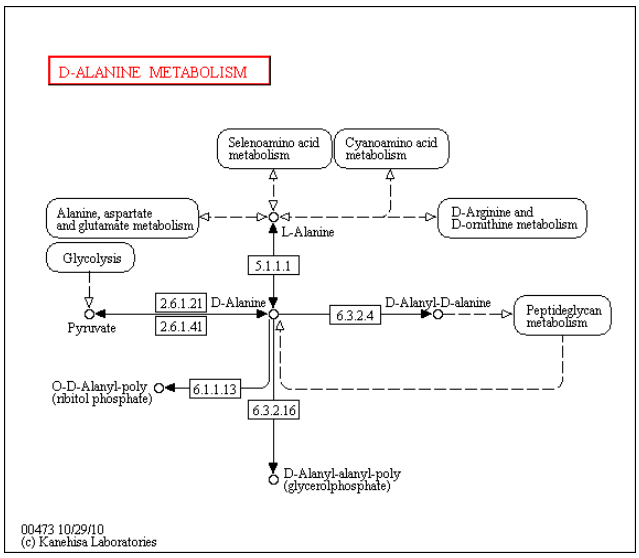

Figura 3.2: Ejemplo de imagen de pathway

Al segundo grupo pertenece:

  – tercera columna (Enzyme): nombre descriptivo de la enzima.

- cuarta columna (EnzymeId): identificador de la enzima.
- quinta columna (NrSeqs): número de secuencias de mRNA que codifican la enzima.
- sexta columna (Seqs): nombre de las secuencias de mRNA que codifican la enzima.

{Pathway} {Seqs in Pathway} {Enzyme} {EzymeId} {NrSeqs} {Seqs} {PathwayId} {PathwayImage}
D-Alanine metabolism 4 D-amino-acid transaminase ec:2.6.1.21 3 Ciclev10008846m, Ciclev10008593m, Ciclev10008940m path:map00473 /home/ralonso/B2G/Naranjoma/Naranjoma/kegg/map00473.gif

Tabla 3.15: Formato del fichero kegg_txt_20110915_1636.txt

Se identifican tres elementos por entrada: pathway, que incluye su identificador y descripción; enzima con identificador y descripción también y una lista de mRNAs. Estos mRNA (y por lo tanto los genes a partir del cual se transcribe) son los encargados de codificar las enzimas las cuales, a su vez, intervienen en los pathways. En cuanto a las cardinalidades de las relaciones, una enzima se relaciona con uno o muchos mRNAs, pathway y genes. Un pathway es afectado por una o más enzimas. Tanto el elemento mRNA como el elemento gen pueden relacionarse con más de una enzima (Tabla 3.16).

Elementos	
Concepto	**Atributos**
pathway	*ID*
	descripción
	secuencias en pathway
	dirección imagen
enzima	*ID*
	descripción
	número de secuencias
mRNA	*nombre*
Relaciones	
Concepto 1	**Concepto 2**
enzima	mRNA
enzima	gen
enzima	pathway

Tabla 3.16: Datos extraídos de ficheros kegg_txt_20110915_1636.txt

VCF

El IVIA trabaja con un gran número de variedades de cítricos. De cada una de ellas existen dos ficheros VCF: un fichero para variaciones de tipo Polimorfismo de Nucleótido Sencillo (SNP) y otro para variaciones de tipo indel. Contienen la lista de variaciones de dicha especie con respecto a su secuencia de referencia. En total se disponen de 80 ficheros con formato VCF.

Estos ficheros siguen el estándar definido para este formato [29][32]. Al inicio del documento se incluye la metainformación del fichero. Estas líneas tienen ## como primeros caracteres y se estructuran mediante pares de clave-valor. Cabe destacar que su aparición en el fichero es opcional excepto la línea referida a la versión. La primera línea indica la versión del estándar que sigue el fichero.

Seguidamente, se incluye información sobre la columna INFO donde se describen cada uno de los atributos que aparecen en esta columna. Seguido del nombre se indica el número y tipo de cada atributo entre corchetes. El tipo del atributo puede ser uno de los siguientes: *Integer*, *Float*, *Flag*, *Character* o *String*. El número del atributo es un entero que indica el número de valores de dicho campo, los valores posibles son: un número entero (por ejemplo, si el valor es uno indica que el atributo contendrá un único valor, si tiene dos el atributo contendrá dos valores, etc.), A (un valor por cada alelo alternativo), R (un valor por cada alelo incluyendo el de referencia), G (un valor por cada genotipo), . (un número indefinido de valores) y 0 (no tiene ningún valor, se utiliza cuando el atributo es de tipo *Flag*):

- **AC (Allelic Count) [A, Integer]**: recuento de los alelos en el genotipo para cada alelo alternativo.
- **AF (Allele Frequency) [A, Float]**: frecuencia alélica para cada alelo alternativo.
- **AN (Allele Number) [1, Integer]**: número total de alelos en el genotipo.
- **BaseQRankSum [1, Float]**: Puntuación Z obtenida de la suma de rangos de Wilcoxon de la calidad de las bases del alelo alternativo frente a la calidad de las bases del alelo de referencia. El resultado ideal es un valor cercano a cero, lo que indica que no hay diferencia o que esta es pequeña. Un valor negativo indica que la calidad de las bases del alelo alternativo es menor a la calidad de las bases del alelo de referencia. Un valor lejano de cero indica que se ha podido producir un error a la hora de la secuenciación
- **DP (Read Depth) [1, Integer]**: profundidad de lectura aproximada.
- **DS (Downsample) [0, Flag]**: indica si alguna muestra fue submuestreada.
- **Dels [1, Float]**: fracción de las lecturas que tienen aparición de deleciones.
- **Fs (Fisher's Test) [1, Float]**: valor de p utilizando la escala Phred usando el test de Fisher [33] para detectar sesgos en la hebra. Un sesgo de cadena es un tipo de sesgo dado cuando, al secuenciar una cadena de ADN una de las hebras se ve favorecida frente a la otra, resultando en una incorrecta evaluación de la cantidad de evidencia observada para un alelo frente a otro. Cuanto mayor sea el valor, mayor será el sesgo.

- **HaplotypeScore [1, Float]**: consistencia del sitio con como máximo dos haplotipos. Para todo organismo diploide se espera que una muestra no tenga más de dos haplotipos segregados. En caso contrario, las lecturas realizadas deben considerarse cuanto menos sospechosas. Valores altos indican regiones con alineamientos erróneos.

- **InbreedingCoeff [1, Float]**: coeficiente de endogamia estimado a partir de las probabilidades de genotipo por muestra en comparación con la expectativa de Hardy-Weinberg. Estima si existe evidencia de endogamia en una población: valores altos indican una alta probabilidad de endogamia.

- **MLEAC [A, Integer]**: expectativa de máxima similitud para el recuento de alelos por cada alelo alternativo (similar pero no tiene por qué tomar el mismo valor que AC).

- **MLEAF [A, Float]**: expectativa de máxima similitud para la frecuencia de alelos por cada alelo alternativo (similar pero no tiene por qué tomar el mismo valor que AF).

- **MQ (Mapping Quality) [1, Float]**: raíz cuadrada de la calidad de mapeo de las lecturas en toda la muestra.

- **MQ0 [1, Integer]**: número de mapeo de lecturas cuyo resultado es cero. Valores altos corresponden con regiones con secuenciaciones poco confiables.

- **MQRankSum [1, Float]**: puntuación Z de la prueba de suma de rangos de Wilcoxon de la calidad de mapeo del alelo alternativo frente a la calidad de mapeo del alelo de referencia. El resultado ideal es cero, lo cual indica que hay poca o ninguna diferencia; valores negativos indican que la calidad de la lectura del alelo alternativo es menos que la del alelo de referencia y viceversa. Este valor se utiliza para evaluar la confianza.

- **QD [1, Float]**: calidad de la variación por profundidad. Pone la puntuación del atributo QUAL en perspectiva normalizándolo para la cobertura disponible. Esto es útil puesto que como cada lectura contribuye al valor de QUAL, pueden haber variaciones que inflen artificialmente el valor de QUAL, dando la impresión de que hay una mayor evidencia de la real.

- **RPA [., Integer]**: número de veces que se repite la secuencia del Short Tandem Repeat por cada alelo.

- **RU (Repeated Units) [1, String]**: bases que forman parte del Short Tandem Repeat

- **ReadPosRankSum [1, Float]**: Puntuación Z del test de la suma de rangos de Wilcoxon del sesgo de posición de lectura del alelo alternativo frente al sesgo de posición de lectura del alelo de referencia.

- **Short Tandem Repeat (STR) [0, Flag]**: Indica si el alelo es un Short Tandem Repeat o no.

También se puede incluir información sobre el campo FILTER, que indica qué filtros han sido aplicados a los datos. Cada línea tiene un par clave-valor con el identificador y un segundo par clave-valor opcional con una descripción. En este caso se han aplicado dos filtros, definidos en la Tabla 3.17.

ID	Descripción
LowQual	Low quality
STD_FILTER	QD <2.0 || ReadPosRankSum <-20.0 || InbreedingCoeff <-0.8 || FS >200.0

Tabla 3.17: Filtros aplicados en los ficheros VCF

Existen campos específicos del genotipo. Estos se encuentran el la columna FORMAT y constan de cuatro pares clave-valor: identificador, número, tipo y descripción. A continuación, se muestra una tabla la metainformación referente al campo FORMAT presente en los ficheros que pueden ser consultados en la Tabla 3.18:

ID	Número	Tipo	Descripción
AD	R	Integer	Profundidad de lectura para los alelos de referencia y alternativos
DP	1	Integer	Profundidad de lectura alélica en esa posición para esa muestra
GQ	1	Integer	Calidad del genotipo en la escala Phred
GT	1	String	Genotipo codificado con valores asignados a los alelos separados por / (no se sabe cual de los dos cromosomas tiene el alelo) o | (se conoce el cromosoma que contiene el alelo). El valor 0 representa el alelo de referencia y los alternativos empiezan en 1 en order incremental.
PL	G	Integer	probabilidades normalizadas en la escala Phred para los genotipos

Tabla 3.18: Valores del campo FORMAT en los ficheros VCF

Por último, es posible añadir una lista de los cóntigos (un cóntigo son elementos de ADN superpuestos que al unirse representan una región). incluyendo su identificador y longitud (en nuestro caso los cóntigos equivalen a los scaffolds, siendo en total mil trescientos noventa y ocho).

Queda patente la importancia de la información presente en los metadatos de estos ficheros. Una vez detallados podemos empezar a analizar los datos en sí mismos; en total existen diez columnas tal y como se detalla a continuación (Tabla 3.19:

– **CHROM**: si bien esta columna es llamada cromosoma, contiene un identificador de los cóntigos definidos en los metadados. En función de los datos

proporcionados observamos que todos los valores empiezan por la cadena de texto “scaffold_” seguido de un número entero de entre uno y cuatro dígitos.

- **POS**: la posición de referencia en base 1. Las posiciones se ordenan numéricamente en orden creciente. Debe tratarse de un número entero
- **ID**: lista de identificadores disponibles para dicha variación en caso de que existan. Al trabajar en un ámbito tan minoritario no hay ninguna variación identificada y en su lugar se muestra el carácter “.”.
- **REF**: conjunto de bases presentes en la secuencia de referencia (A, C, G, T, N). Múltiples bases están permitidas. En el caso de inserciones donde este valor sería vacío se incluye la base anterior.
- **ALT**: conjunto de bases separadas por comas de los alelos alternativos, las bases pueden ser: A, C, G, T, N o *. El carácter * está reservado para indicar que el alelo no está presente debido a un borrado superpuesto. En el caso de deleciones donde este valor sería vacío se incluye la base anterior
- **QUAL**: valor en la escala de Phred de la calidad de los alelos alternativos.
- **FILTER**: estatus del filtro. Indica si la variación ha pasado o no los filtros de calidad.
- **INFO**: información adicional presente para la variación. Pares de elementos clave-valor (información detallada anteriormente cuando se explicaron los metadatos de los ficheros, separados por punto y coma.
- **FORMAT**: información referente al genotipo. Se trata del formato especificado para los datos, incluyendo el tipo de datos y el orden. El primer campo debe ser siempre GT (el genotipo). Existen muchos campos definidos, pero en nuestro conjunto de datos tan sólo tenemos cuatro campos (véase la Tabla 3.18).
- **VALUE**: valores de los campos definidos en la columna format.

{CHROM} {POS} {ID} {REF} {ALT} {QUAL} {FILTER} {INFO} {FORMAT} {value}
scaffold_1 7275 . T TG 830.73 PASS AC=2;AF=1.00;AN=2;DP=23;FS=0.000;MLEAC=2;MLEAF=1.00;MQ=27.28; MQ0=0;QD=28.07;RPA=3,4;RU=G;STR GT:AD:DP:GQ:PL 1/1:0,22:23:66:868,66,0

Tabla 3.19: Formato de fichero con formator vcf

En relación a los atributos obtenidos, es necesario realizar varias puntualizaciones: aunque no estén en los datos, es posible obtener qué tipo de variación es y

la variedad de cítrico a la que pertenece gracias al nombre de los ficheros (concatenación de estos dos datos). Ha sido necesario definir un proceso de generación de identificadores para la generación del campo ID. Durante las diversas reuniones realizadas con el IVIA se ha definido que el identificador de una variación será la concatenación del tipo de variación, el *scaffold*, la posición y los alelos de referencia y alternativos. La variedad se ha obviado ya que una misma variación puede estar en varias variedades (Tabla 3.20).

Elementos	
Concepto	**Atributos**
variación	*ID* *POS* *ALT* *REF* *QUAL* *FILTER* *INFO (AC, AF, AN, BaseQRankSum, DP, DS, Dels, Fs, HaplotypeScore, InbreedingCoeff, MLEAC, MLEAF, MQ, MQ0, MQRankSum, QD, RPA, RU, ReadPosRankSum, STR)* *FORMAT-VALUE (GT,AD,DP,GQ,PL)* *variety* *type*
Relaciones	
Concepto 1	**Concepto 2**
variación	scaffold

Tabla 3.20: Datos extraídos de ficheros vcf

SnpEff

Tras analizar y extraer la información relevante de los datos inicialmente disponibles. El siguiente paso ha consistido en anotar los ficheros VCF utilizando la herramienta SnpEff[7]. Esta herramienta añade anotaciones a las variaciones; estas anotaciones predicen el efecto a nivel funcional de dicha variación [34]. Partiendo de un conjunto de variaciones, típicamente un fichero en formato VCF, produce un nuevo fichero añadiendo anotaciones calculando el efecto que producen en los genes. Este fichero añade las cabeceras necesarias al fichero con la información sobre los nuevos campos tal y como lo requiere el estándar de VCF.

Para producir las anotaciones la herramienta necesita una base de datos. En esta base de datos puede ser descargada del catálogo de bases de datos existentes o ser construida utilizando un conjunto definido de ficheros. A pesar de disponer de veinte mil genomas de referencia, el genoma de la mandarina, usada como

[7] http://snpeff.sourceforge.net/.

secuencia de referencia en nuestro caso, no estaba disponible por lo que se generó la base de datos correspondiente con el objetivo de anotar nuestros VCF.

El primer paso en la generación de la base de datos es modificar el fichero de configuración de la herramienta llamado `snpEff.config`. En este fichero se almacenan referencias a todas las bases de datos disponibles, nombres de directorios, tablas de codones, etc. Se define el nombre de la nueva base de datos (véase Fragmento de código A.9). A continuación, se crea el directorio `data` en el directorio de la herramienta. Dentro de este directorio se crean dos carpetas más: `genomes` donde se almacenará la secuencia de referencia (`Cclementina.fa`) y `Cclementina` (nombre añadido al fichero de configuración) donde se almacenarán los ficheros de anotación y auxiliares (cds.fa, genes.fa, protein.fa, transcript.fa). En la Tabla 3.21 se pueden ver los ficheros y su función:

Fichero	**Atributo**
Cclementina.fa	Secuencia de referencia. Fichero original: Cclementina_v1.0_scaffolds.fa
cds.fa	Comprobación de que los CDS predichos por el fichero de genes. Fichero original: Cclementina_182_v1.0.cds.fa concuerdan con los reales
genes.fa	Información de los genes anotados y generación de región codificantes. Fichero original: Cclementina_182_v1.0.gene_exons.gff3
protein.fa	Obtención de las anotaciones de tipo *rare amino acids*[8] y comprobación de que las proteínas predichas por las secuencias corresponden con las reales. Fichero original: Cclementina_182_v1.0.protein.fa
transcript.fa	Se utiliza para realizar comprobaciones adicionales de seguridad. Fichero original: Cclementina_182_v1.0.transcript.fa

Tabla 3.21: Ficheros utilizados por SnpEff

Tras preparar los ficheros necesarios sólo queda ejecutar la herramienta para que produzca los ficheros VCF anotados. Para ello se ejecuta el programa para cada uno de los ficheros. La orden de ejecución del programa incluyendo los parámetros necesarios se puede observar en el Fragmento de código A.1 . La herramienta se ejecuta mediante un fichero ejecutable en formato Java ARchive (JAR). La ejecución de la herramienta precisa cuatro argumentos:

- `memoria`: indica la máxima memoria que lanzará el ejecutable
- `base_de_datos`: identifica el directorio en el cual se encuentran los datos (Cclementina en nuestro caso)
- `ruta_al_fichero_de_entrada`: indica al programa qué fichero VCF debe procesar

[8] aminoácidos que no se incorporan a proteínas.

– `ruta_al_fichero_de_salida`: proporciona nombre y ruta del fichero anotado generado por la herramienta

De este modo, en el Fragmento de código A.2 se incluye una parte del script (en el script completo es una entrada por cada fichero). Para este ejemplo se han escogido las variaciones de tipo indel y Polimorfismo de Nucleótido Sencillo de la variedad 000 para que la herramienta las anote y genere los ficheros VCF anotados correspondientes.

Vcf anotados

Tras la ejecución de la herramienta encontramos una gran cantidad de nuevos datos; de media cada fichero ha triplicado su tamaño original pasando de 30GB a 90GB. Estos nuevos datos deben ser procesados y analizados para poder agregarlos correctamente al esquema conceptual. Como hemos indicado anteriormente, los campos de las anotaciones de las variaciones se han agregado como metadatos a los ficheros. Existen tres tipos de anotaciones generadas:

– ANN: anotaciones funcionales de la variación.

– LOF: anotaciones que predicen pérdida de funcionalidad causada por la variación.

– NMD: anotaciones que predicen la decadencia media del mRNA del gen causada por la variación.

```
1  ##reference=file:///clinicfs/projects/naranjoma/pipeline/re sources/citrus_clementina.fasta
2  ##SnpEffVersion="4.3r (build 2017-09-06 16:41), by Pablo Cingolani"
3  ##SnpEffCmd="SnpEff  Cclementina ~/ivia_000_snp.vcf "
4  ##INFO=<ID=ANN,Number=.,Type=String,Description="Functional annotations: 'Allele |
     ↪  Annotation | Annotation_Impact | Gene_Name | Gene_ID | Feature_Type | Feature_ID |
     ↪  Transcript_BioType | Rank | HGVS.c | HGVS.p | cDNA.pos / cDNA.length | CDS.pos /
     ↪  CDS.length | AA.pos / AA.length | Distance | ERRORS / WARNINGS / INFO' ">
5  ##INFO=<ID=LOF,Number=.,Type=String,Description="Predicted loss of function effects for
     ↪  this variant. Format: 'Gene_Name | Gene_ID | Number_of_transcripts_in_gene |
     ↪  Percent_of_transcripts_affected'">
6  ##INFO=<ID=NMD,Number=.,Type=String,Description="Predicted nonsense mediated decay effects
     ↪  for this variant. Format: 'Gene_Name | Gene_ID | Number_of_transcripts_in_gene |
     ↪  Percent_of_transcripts_affected'">
```

Figura 3.3: Cabecera de metadatos en ficheros VCF anotados

En los metadatos de los ficheros VCF (Figura 3.3) se han incluido los posibles atributos que estas anotaciones contienen (véase la Tabla 3.22). Las anotaciones de tipo ANN poseen dieciséis campos mientras que las anotaciones de tipo LOF y NMD (comparten los mismos campos) cuatro. Estas anotaciones añaden un mayor nivel de información acerca de la variación que sólo las coordenadas cromosómicas. ¿Está la variación en un gen? ¿En un exón? ¿Modifica la codificación de una proteína? ¿Genera un codón de terminación prematuramente? Las anotaciones ayudan a responder todas estas preguntas. Estas anotaciones varían enormemente en su grado de complejidad, siendo las más complejas las que más

dependen de las predicciones del algoritmo de la herramienta y estas predicciones pueden ser erróneas.

{CHROM} {POS} {ID} {REF} {ALT} {QUAL} {FILTER} {INFO};{Annotations} {FORMAT} {value}
scaffold_1 595 . C G 173.77 PASS AC=1;AF=0.500;AN=2;BaseQRankSum=-1.189;DP=20;Dels=0.00;FS=0.000; HaplotypeScore=0.0000;MLEAC=1;MLEAF=0.500;MQ=47.71;MQ0=0; MQRankSum=-3.090;QD=8.69;ReadPosRankSum=1.268;ANN=G|intergenic_region|MODIFIER|CHR_START-Ciclev10010164m.g|CHR_START-Ciclev10010164m.g.v1.0|intergenic_region|CHR_START-Ciclev10010164m.g.v1.0|||n.595C>G||||| GT:AD:DP:GQ:PL 0/1:13,7:19:99:202,0,407

Tabla 3.22: Formato de fichero con formato VCF anotados con la herramienta SnpEff

Empezando por las anotaciones de tipo ANN, estas tienen dieciséis atributos como se puede observar en la Tabla 3.23.

Nombre	**Descripción**
Allele	Identifica el alelo al cual se refiere la anotación (ayuda cuando existe más de un alelo)
Annotation	Efecto de la anotación utilizando términos de *Gene Ontology*. Múltiples efectos pueden ser concatenados con el carácter "&"
Annotation_impact	Estimación del impacto de la anotación (HIGH, MODERATE, LOW, MODIFIER)
Gene_Name	Nombre del gen
Gene_ID	Identificador del gen
Feature_type	Indica el tipo de característica o rasgo del siguiente campo (tránscrito, motif, micro ARN, etc. aunque en nuestro caso los valores posibles son transcript, intergenic_region o gene_variant)
Feature_ID	identificador de la característica
Transcript_Biotype	si el campo Feature_ID tiene el biotipo transcript si está disponible
Rank	indica los exones e intrones que son *rank* y el total de exones e intrones
HGVS.c	variación utilizando notación HGVS
HGVS.p	variación utilizando notación HGVS a nivel proteico si la variación es codificante

Continúa en la página posterior

Continúa de la página anterior

Nombre	Descripción
cDNA.pos/cDNA.length	posición en el ADNc [9] y la longitud de este
CDS.pos/CDS.length	posición y número de bases codificantes
AA.pos/AA.length	posición y número de bases AA (incluyendo el codón de inicio, pero no el de fin)
Distance	distancia a la característica
ERRORS/WARNINGS/INFO	información adicional que puede afectar a la precisión de la anotación

Tabla 3.23: Atributos de la anotación ANN

Se ha identificado una nueva entidad: ANN. Cada variación de los ficheros puede tener muchas anotaciones de tipo ANN, por lo que por cada una se debe generar una nueva entidad de tipo ANN que tendrá los dieciséis atributos expuestos anteriormente y, dependiendo de la localización de la variación, relaciones con: el gen en el cual se encuentra la variación (en el caso de que la variación sea intergénica se relacionará con los dos genes entre los cuales se encuentra) y el mRNA donde se encuentra la variación (Tabla 3.24).

Elementos	
Concepto	**Atributos**
ANN	*Allele* *Annotation* *Annotation_Impact* *Gene_Name* *Gene_ID* *Feature_Type* *Feature_ID* *Transcript_Biotype* *Rank* *HGVS.c* *HGVS.p* *cDNA.pos/cDNA.length* *CDS.pos/CDS.length* *AA.pos/AA.length* *Distance* *errors/warnings/info*
Relaciones	
Concepto 1	**Concepto 2**
gen	ANN

Continúa en la página posterior

[9] hebra de ADN de doble cadena una de las cuales constituye una secuencia totalmente complementaria del ARN mensajero a partir del cual se ha sintetizado.

Concepto	Atributos
mRNA	ANN

Tabla 3.24: Datos extraídos de anotaciones de tipo ANN en ficheros VCF anotados

Por otro lado, las anotaciones de tipo LOF y NMD comparten los mismos atributos y son mucho menos comunes (mil sesiscientas veces más ANN que LOF y seis mil quinientas veces más ANN que NMD). Estos tipos de anotaciones presentan cuatro atributos como se puede ver en la Tabla 3.25

Nombre	Descripción
Gene	Nombre del gen al que afecta la anotación
ID	Identificador del gen al que afecta la anotación
num_transcripts	Número de tránscritos en el gen
percent_affected	porcentaje de tránscritos afectados por la variación

Tabla 3.25: Atributos de la anotación LOF y NMD

Se han identificado dos nuevas entidades: LOF y NMD. Cada variación de los ficheros VCF puede tener varias anotaciones de cualquiera de los dos tipos. Así pues, por cada una se genera una nueva entidad con sus cuatro atributos. Cada entidad de estos dos tipos se relaciona con la variación a la que pertenece y el gen al que afecta (las anotaciones no proporcionan información sobre qué tránscritos afectan en caso de no ser el cien por cien por lo que no se añade la relación entre las entidades LOF y NMD y los mRNAs) (Tabla 3.26).

Elementos	
Concepto	**Atributos**
LOF	*Gene*
	ID
	num_transcripts
	percent_affeceted
NMD	*Gene*
	ID
	num_transcripts
	percent_affeceted
Relaciones	
Concepto 1	**Concepto 2**
gen	LOF
mRNA	LOF

Continúa en la página posterior

Continúa de la página anterior

Concepto	Atributos
gen	NMD
mRNA	NMD

Tabla 3.26: Datos extraídos de anotaciones de tipo LOF y NMD en ficheros VCF anotados

De todos estos ficheros en total se han identificado diecisiete entidades y veintiocho relaciones entre ellas. A continuación, se detallarán las decisiones que han llevado al ECGCit. de la Tabla 3.4. Se trata de una de las dos contribuciones esenciales de este trabajo. Paso necesario y fundamental para asentar el conocimiento obtenido tras el proceso de análisis de los ficheros disponibles. Sientan las bases de nuestro conocimiento y dirige el proceso ETL ya que es una guía y punto de consulta ante duda. La dispersión y heterogeneidad de los ficheros analizados (169 ficheros) suma si cabe todavía más valor a este esquema ya que ha conseguido homogeneizar y unir toda esta información.

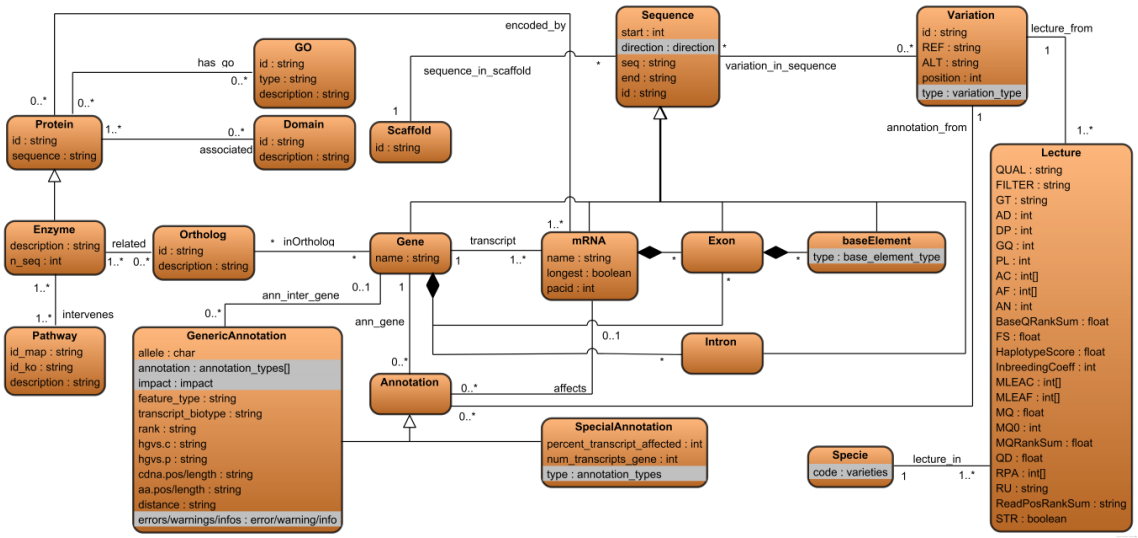

Figura 3.4: Esquema Conceptual de los Cítricos (ECGit.)

Debido a su naturaleza algunos atributos presentan tipos enumerados, en la Figura 3.5 se puede ver los distintos valores que estos campos toman:

- **baseElement**: `type`
- **GenericAnnotation**: `annotation`, `impact`, `errors/warnings/info`
- **Specie**: `code`
- **Sequence**: `direction`
- **SpecialAnnotation**: `type`
- **Variation**: `type`

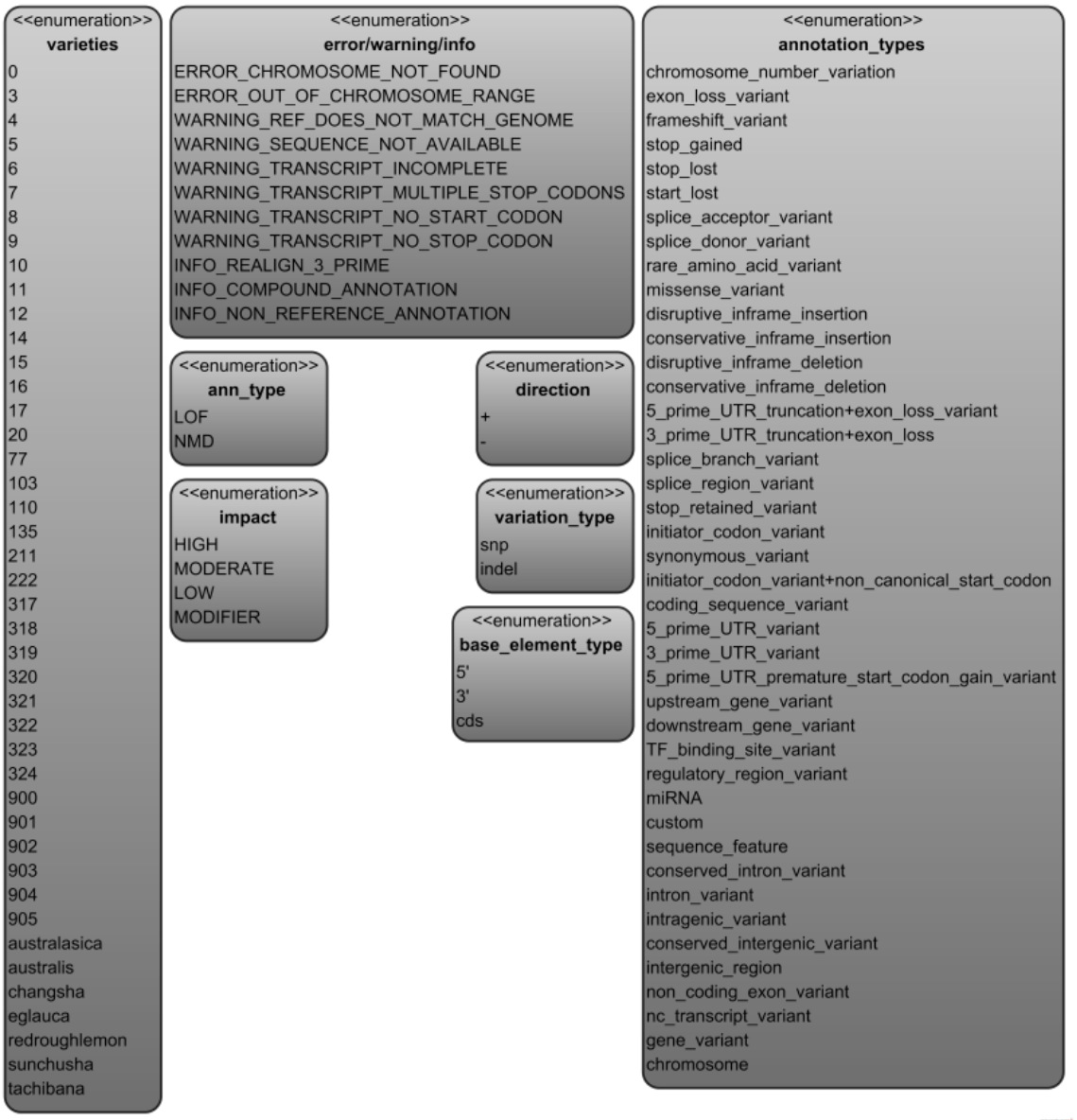

Figura 3.5: Enumerados de los datos del ECGCit.

Se han tomado una serie de decisiones de diseño que pasan a explicarse a continuación. En primer lugar, se ha decidido extraer el atributo secuencia obtenido en la fase de análisis de ficheros como una nueva entidad. Con este cambio, un *scaffold* pasa a estar formado de varias secuencias (siguiendo su definición). El hecho de extraer la secuencia de los elementos nos permite una mayor flexibilidad al poder "romper" una secuencia en múltiples subsecuencias. Estas subsecuencias se pueden, o no, hacer coincidir con las posiciones del resto de elementos.

En segundo lugar, era necesario organizar y estructurar de una manera coherente los elementos cromosómicos existentes en los ficheros. Como se ha explicado anteriormente, estos siguen una estructura jerárquica de más general a menos, es decir, que siguen una relación del tipo "está en". Por este motivo, cuanto menos general sea un elemento (es decir, más abajo en la jerarquía) menor será la longitud de su secuencia. Precisamente, el concepto que se repite con todos los elementos es el de secuencia. Se ha aprovechado este hecho para hacer que el res-

to de elementos hereden de la entidad *Sequence*. Esta nueva clase *Sequence* corresponde a una secuencia arbitraria de nucleótidos. Posee los siguientes atributos:

- *direction*: dirección de la cadena: de 5′ a 3′ o viceversa.
- *start*: posición de inicio de la secuencia.
- *seq*: secuencia de nucleótidos.
- *end*: posición de fin de la secuencia.
- *id*: identificador único para la secuencia definida.

Así pues, partiendo de la información de los ficheros, una secuencia puede ser un gen, un mRNA, un exón, un CDS, un UTR 3′ o un 5′ cap. La mayoría de atributos de estos elementos han sido movidos a la nueva entidad *Sequence* a excepción del nombre del gen y mRNA y el atributo *longest* de este último. Se ha excluido el atributo *transcript* del mRNA por ser redundante. El atributo *Parent* ha sido eliminado también puesto que su valor está contenido en las relaciones de composición entre los elementos. El atributo *pacid*, presente en todos los elementos a excepción del gen, se ha mantenido tan sólo en el elemento mRNA puesto que es un identificador numérico de este elemento. Los elementos CDS, UTR 3′ y 5′ cap se han unido en una sola entidad llamada *BaseElement* con un atributo llamado *type* para saber qué tipo de elementos de los tres es. Se ha tomado esta decisión porque, jerárquicamente, los tres elementos están al mismo nivel. Además de las entidades identificadas en los ficheros, se ha incluido uno nuevo: intrón. En un gen se encuentran secuencias en exones e intrones; teniendo los primeros identificados es posible derivar los segundos.

En segundo lugar, se ha creado una relación de herencia entre las clases *Protein* y *Enzyme* ya que, conceptualmente, una enzima es un tipo de proteína. El atributo *annot-version* se ha eliminado porque se trabaja únicamente con una versión y esta no es relevante para nuestro caso de uso concreto. Los atributos de "secuencias en pathway" y la dirección de su imagen se han eliminado porque son atributos que quedan fuera del ámbito de nuestro caso de uso.

En tercer lugar, los tres tipos de anotaciones que se han identificado pueden agruparse en dos grupos: GenericAnnotation (campo ANN) y SpecialAnnotation (campos LOF y NMD que comparten atributos). Estas dos clases heredan, a su vez de la clase Annotation. Esta herencia permite simplificar el modelo a la hora de establecer las relaciones entre las clases Annotation, Gene y Variation. Para los casos de las anotaciones intergénicas, se utiliza la relación adicional ann_inter_gene para el segundo gen.

En cuarto y último lugar, las variacion se han dividido en tres clases: *Variation*, *Lecture* y *Specie*. Nótese que una misma variación puede estar presente en múltiples especies de cítricos y cada una de estas apariciones tener parámetros de calidad de secuenciación diferentes. Debido a esto, la clase *Variation* contiene la información necesaria para identificar una variación: alelo de referencia, alelo alternativo, posición de la variación y tipo de variación. También es la clase que se relaciona con las anotaciones (son las mismas, aunque se secuencien especies diferentes porque la secuencia de referencia es la misma) y las secuencias

correspondientes. La clase *Lecture* tan sólo contiene los parámetros de calidad de secuenciación de la misma. La clase *Specie* identifica la variedad a la que pertenece una lectura.

El resto de entidades y relaciones identificadas se han respetado en la generación del esquema.

3.2 Especificación de Requisitos

En este capítulo se realiza la especificación de requisitos. Es esencial identificar cuáles son los objetivos necesarios para conseguir un tratamiento efectivo y eficiente de la información del genoma de los cítricos estudiado. La complejidad de la información implicada en esa gestión avanzada de datos hace que esta fase de especificación de requisitos sea particularmente esencial.

Muchas veces los requisitos no son evidentes sin un minucioso análisis. En nuestro contexto, este hecho se ve agravado por la enorme complejidad del dominio y de los objetivos de los *stakeholders*. Esta complejidad se traduce en la necesidad de colaboración entre los *stakeholders*, que poseen el conocimiento del dominio pero quizás no el conocimiento técnico necesario para explicitar las necesidades y los investigadores, con el conocimiento técnico necesario para la consecución del proyecto. De la necesaria colaboración entre ambos surgen los requisitos de un proyecto regido por la metodología de *Design Science*. En esta metodología, los requisitos se definen como propiedades del tratamiento deseado. La deseabilidad de un requisito debe venir motivada en términos de cómo contribuye a los objetivos (Sección 1.2) de los *stakeholders* (Sección 2.4) mediante argumentos de contribución (esos argumentos de contribución son razones dadas por las que un artefacto que cumpla los requisitos definidos ayudará a alcanzar los objetivos del stakeholder en el contexto del problema).

R0 Conocer el grado de diferenciación entre grupos de variedades

Se parte de dos grupos arbitrarios de variedades definidas por el usuario. Se estudiarán las variaciones pertenecientes a estos dos grupos de variedades. A estas variaciones se le aplicarán restricciones de calidad, profundidad, flexibilidad e impacto. Inicialmente, se trabaja con las variaciones que están presentes en todas las variedades del primer grupo y en ninguna del segundo grupo. Es decir, trabajamos con la diferencia entre la intersección del primer conjunto y la unión del segundo. Siendo A y B grupos de variedades:

Let $C_A = \forall a \in A, x \in a$
Let $C_B = \exists b \in B, x \in b$
Let $C_f = C_A \setminus C_B$

Cuando se realiza la secuenciación de las variedades, estas vienen acompañadas de una serie de parámetros de calidad. Estos parámetros de calidad son interesantes puesto que permiten refinar las variaciones que se aceptan utilizan-

do filtros de calidad más o menos estrictos. Se utilizan tres atributos para el filtro de calidad:

- Profundidad de lectura aproximada (DP): si la variación tiene un valor DP por debajo del especificado no se tiene en cuenta (parámetro de calidad basado en las lecturas realizadas durante el proceso de secuenciación).
- Calidad del genotipo en la escala de Phred (GQ): si la variación tiene un valor GQ por debajo del especificado no se tiene en cuenta.

Un ejemplo de filtro de calidad sería que, para aceptar una variación, esta tenga un valor DP superior a 70 y un valor GQ superior a 10.

Otro atributo a tener en cuenta es la profundidad de lectura para los alelos de referencia y alternativos (AD). Se genera un valor derivado (frecuencia alélica) a partir de estos campos. Este valor derivado (frecuencia alélica) para el campo AD se calcula como sigue (sabiendo que el campo AD está formado por dos valores):

1. Si el valor del segundo campo es cero, la frecuencia alélica es cero.
2. Si el valor del primer campo es uno, la frecuencia alélica es uno.
3. En cualquier otro caso, se le asigna el valor resultante de la siguiente operación:

$$\frac{AD[1]}{AD[1] + AD[0]}$$

Se pueden establecer distintas condiciones de rango para este valor, de modo que se filtren aquellas variaciones que presenten un valor mayor o menor a uno dado o que se encuentren dentro de un rango concreto dependiendo del tipo de análisis. Estas condiciones pueden componerse y establecer que cumplan una condición de un conjunto dado.

Respecto a la profundidad posicional, se desea poder filtrar las variaciones resultantes según su posición dentro del genoma. Las variaciones deben poderse filtrar según si se encuentran fuera o dentro de un gen, dentro de un mRNA, de un exón, de un intrón, de un CDS o de una región UTR.

En ocasiones, y por fallos en las lecturas de variaciones, se pueden producir falsos positivos o falsos negativos. Para lidiar con estos eventos, se debe poder aplicar un límite de flexibilidad que permita tener en cuenta variaciones que aparezcan en casi todas las variedades del primer grupo o que aparezca en un número muy reducido de variedades del segundo. Por ejemplo, si tenemos dos grupos de cinco variedades, añadir la flexibilidad para mostrar las variaciones que aparezcan en cuatro o más variedades del primer conjunto y que aparezcan como mucho en una variedad del segundo conjunto.

Por último, también se debe poder filtrar por el impacto que una variación ejerce. No tiene la misma importancia una variación que provoca la creación de proteínas no funcionales y otra cuyo efecto es inocuo. Por tanto, se podrán filtrar las variaciones según el impacto que produzcan.

Al aplicar todos estos filtros se obtendrán un conjunto de variaciones. Estas variaciones pueden afectar a genes y, a su vez, estos genes pueden intervenir en la creación de proteína, enzimas, formar parte de dominios, etc. Todos estos datos deben ser proporcionados por cada una de las variaciones.

R1 Conocer el grado de diferenciación entre grupos de variedades filtrando por genes

Se ha iterado sobre el requisito R0 para obtener este requisito. Radica en una actualización del mismo, pero modificándolo para que acepte una nuevo parámetro de búsqueda. El nuevo parámetro consiste en una lista de genes definidos por el usuario. Estos genes poseen una posición de inicio y de fin dentro del *scaffold* al que pertenece. Estas posiciones se utilizan para filtrar las variaciones de modos que tan sólo se incluyan aquellas que posicionalmente estén dentro de los genes definidos.

Adicionalmente, es posible indicar un número de bases arbitrario. Si una variación no se encuentra dentro del rango posicional de un gen, pero sí a una distancia menor que el número de bases definido también se tendrá en cuenta. De este modo se puede relajar la consulta e incluir variaciones que de otro modo no serían tomadas en cuenta.

Tomando como referencia el requisito R0, se elimina la posibilidad de filtrar por profundidad posicional ya que ya se están tomando en cuenta aquellas variaciones que se encuentran posicionalmente dentro de genes o en posiciones cercanas.

R2 Conocer las variaciones existentes en una región cromosómica

Dentro del genoma y, más concretamente, los *scaffolds*, existen regiones que son de especial interés dentro del estudio genómico. Por lo tanto, debe ser posible buscar variaciones dentro rangos en regiones cromosómicas. Además de la región cromosómica, las variaciones, al igual que con el resto de consultas, deben poder ser filtradas según el valor de los atributos DP, GQ y AB, así como el impacto que realiza sobre el gen al que afecta.

Como se observa en la definición de los requisitos, el objetivo final de las consultas es determinar qué grado de diferenciación existe entre distintos grupos de variedades y cómo estas diferencias se expresan en el genotipo. Esta diferenciación puede obtenerse utilizando grupos de variedades, conjuntos de genes con zonas adyacentes o regiones cromosómicas. Este proceso realizado de manera manual es tremendamente complejo, lento y especialmente propenso a errores. La implementación de una herramienta que permita automatizar estas tareas de análisis presentando una interfaz clara y usable permitirá a los expertos incrementar su productividad y su eficiencia en varios órdenes de magnitud. Esto se alinea al objetivo del IVIA, cuyo objetivo es aumentar su eficiencia y rapidez. Por un lado, una herramienta automatizada, parametrizada y libre de errores humanos incrementa la eficiencia de los análisis y la calidad de los mismos. Por otro lado, la rapidez aumentará drásticamente si los ejercicios de obtención, filtrado

y comparación de conjuntos de datos se realizan sin necesidad de intervención humana.

3.3 Base de Datos

Esta sección está dividida en cinco subsecciones. La primera subsección justifica el SGBD utilizado en nuestra solución explicando el proceso seguido para su selección (Subsección 3.3.1). Una vez seleccionado el SGBD, la segunda subsección introduce dicho sistema enumerando su funcionamiento interno y características y el proceso de configuración seguido (Subsección 3.3.3). A continuación, en la tercera subsección (Subsección 3.3.4), se detalla el desarrollo del proceso de transformación de los datos (proceso ETL). En la cuarta subsección se realiza la carga de la información en la base de datos Neo4J (Subsección 3.3.5). Finalmente, en la quinta subsección, se muestran los parámetros de configuración y los valores elegidos para el correcto funcionamiento de la base de datos (Subsección 3.3.6).

3.3.1. Selección

La selección de una base de datos adecuada puede, en el mejor de los casos, marcar la diferencia entre un sistema rápido y escalable y uno poco eficiente y, en el peor, suponer la diferencia entre un proyecto exitoso y otro fallido. Por este motivo hemos dedicado este apartado a la selección de la tecnología de bases de datos adecuada para nuestro dominio a fin de asegurar una aplicación que ayude a cumplir nuestros objetivos.

A la hora de seleccionar la tecnología a utilizar para almacenar nuestros datos, el primer interrogante que surge es elegir qué tipo de base de datos utilizar. El primer paso es plantearse qué SGBD utilizar. Podemos elegir entre un Sistema de Gestión de Bases de Datos Relacionales (SGBDR) o un SGBD NoSQL.

Las bases de datos relacionales son aquellas que siguen el *modelo relacional*; estos sistemas siguen un conjunto definido de reglas [35] llamadas *las 12 Reglas de Codd* (empezando por cero):

- Regla original: para todo SGBDR, el sistema debe gestionar las bases de datos enteramente mediante sus capacidades relacionales.

- Regla de la información: toda la información debe estar representada explícitamente en el esquema lógico (los datos están en tablas). Esto implica que incluso los nombres de tablas, columnas y otra metainformación (el catálogo del sistema o diccionario de datos) se almacena en el sistema mediante estructuras relacionales.

- Regla del acceso garantizado: para todos los datos existentes, se cumple que son accesibles a nivel lógico utilizando una combinación de nombre de tabla, valor de clave primaria y nombre de columna.

- Tratamiento sistemático de valores nulos: los valores nulos se utilizan para representar información desconocida o no aplicable con independencia del tipo de datos.

- Diccionario dinámico basado en el modelo relacional: la descripción de la base de datos (metadatos) se representa a nivel lógico de igual forma que los datos normales (usando el modelo relacional).

- Regla del sublenguaje de datos completo: debe existir un lenguaje que soporte la definición de datos, definición de vistas, manipulación de datos y que sea limitante de integridad y de transacción (un ejemplo de este lenguaje sería el Lenguaje de Consulta Estructurada (SQL)).

- Regla de actualización de vistas: todas las vistas teóricamente actualizables se deben actualizar por el sistema.

- Inserción, actualización y borrado de alto nivel: una relación base o derivada se maneja como un solo operando en operaciones de consulta, inserción, actualización y borrado

- Independencia física de datos: Los programas de aplicación permanecen inalterados ante la realización de cambios en las representaciones de almacenamiento o métodos de acceso a nivel físico.

- Independencia lógica de datos: Los programas de aplicación permanecen inalterados ante la realización de cambios en las tablas del modelo a nivel lógico.

- Independencia de integridad: las limitaciones de integridad deben ser especificables en el lenguaje de datos y almacenarse en la base de datos con independencia de los programas de aplicación.

- Independencia de distribución: que la base de datos esté distribuida o no debe resultar transparente.

- Regla de no subversión: si el sistema relacional posee un lenguaje de bajo nivel (un único registro a la vez, también conocido como lenguaje navegacional), este lenguaje no puede ser usado para ignorar o subvertir las limitaciones de integridad definidas en el lenguaje de alto nivel (SQL)

En función de estas reglas, el esquema relacional que utilizan los SGBDRs es rígido. Dependiendo del dominio de los datos, esto puede ser un problema al limitar flexibilidad a la hora de modificar el modelo. Por otra parte, observamos que el procesamiento de las transacciones en los SGBDRs garantiza Atomicidad, Consistencia, Integridad y Durabilidad (ACID). Este principio es una de sus piedras angulares y ofrece una serie de garantías cuando se trabaja con sistemas relacionales [36]:

- Atomicidad: las transacciones se procesan tal que, cuando estas constan de varios pasos, o bien se ejecutan todos o no se ejecuta ninguno: si un paso

falla, la transacción entera fallará y el estado de la base de datos no será modificado. De este modo, las transacciones realizadas exitosamente se presentan como indivisibles (atómicas) mientras que las transacciones no exitosas sencillamente no suceden.

- Consistencia: una base de datos debe presentar un estado válido en todo momento (integridad de los datos de acuerdo a las reglas definidas para dicha base de datos). Esta propiedad garantiza que, partiendo de una base de datos cuyo estado es válido, tras la ejecución de una transacción el estado de la base de datos seguirá siendo válido.

- Aislamiento: las actualizaciones en la base de datos provocadas por el procesamiento de una transacción no deben ser visibles por tras transacciones hasta que la primera sea confirmada. Esta propiedad, cuando se hace cumplir estrictamente, resuelve el problema de actualización temporal y hace innecesario las anulaciones en cascada de las transacciones.

- Durabilidad: los cambios realizados en la base de datos por una transacción confirmada deben ser permanentes y no se pueden perder por ningún motivo.

Al ser una tecnología tan madura que ha sido empleada durante décadas en entornos de producción de tipo Procesamiento de Transacciones en Línea (OLTP), ofrecen una gran velocidad de procesamiento y consistencia. Hay una gran documentación disponible y ya ha tratado o mitigado los problemas que han surgido con el paso del tiempo.

Sin embargo, estos sistemas presentan ciertas limitaciones. En aplicaciones que utilizan datos de manera intensiva su eficiencia es reducida, debido en parte al hecho de que garanticen ACID. La razón de esto es que para garantizar ACID se produce un sobrecoste computacional, así como posibles bloqueos a la hora de ejecutar transacciones (una transacción puede requerir múltiples bloqueos de datos) que afectan negativamente al rendimiento. Toda esta complejidad añadida dificulta la escalabilidad de estos sistemas.

Se observa otra limitación cuando el objeto de interés son las relaciones de los datos por encima de los datos en sí mismos; cuanto mayor sea la complejidad e interconexión de los datos, más le costará a un sistema relacional procesar peticiones en dicha dirección. Esto es debido a la manera en la que se almacenan físicamente las relaciones entre las entidades (referencias por valor de un identificador).

Por otro lado, existen los SGBD NoSQL. Su aparición es relativamente reciente y no siguen un modelo relacional. Existen multitud de tipos y en su mayoría surgen de una necesidad de los desarrolladores de poder utilizar esquemas de información más flexibles en entornos muy cambiantes y con un tiempo de desarrollo reducido. Mención especial al hecho de que *la gran mayoría* de sistemas NoSQL no garantizan ACID.

Existen diversos tipos de SGBD NoSQL[37]:

- Clave-Valor: se trata del modelo más simple. Permiten almacenar información en forma de datos que siguen una estructura de clave valor. La clave

está compuesta de una cadena de texto y el valor contiene la información que se va a almacenar. Al utilizar una estructura similar a las tablas Hash permiten una alta concurrencia y escalabilidad, además de permitir realizar búsquedas (únicamente por clave) muy rápidamente y almacenamiento masivo. Su modelo de almacenamiento fuerza a mantener una estructura de datos extremadamente simple. Se trata del modelo más simple de implementar. Un ejemplo de estos sistemas es DynamoDB[38].

- Orientadas a familias de Columnas: esta familia de bases de datos NoSQL almacena información utilizando una estructura de datos distribuida de columnas permitiendo múltiples atributos por clave. Estas columnas se organizan por familias de columnas. Un ejemplo de estos sistemas es Apache Cassandra [39].

- Orientadas a documentos: diseñadas para almacenar documentos, almacenan su información en formato eXtensible Markup Language (XML) o JavaScript Object Notation (JSON). Siguen un modelo un poco más complejo que los Clave-Valor, pues su campo valor puede contener información semi-estructurada y se puede buscar tanto por clave como por valor. Es especialmente útil para almacenar colecciones de información documental como mensajes, información muy dispersa o información a mostrar en una página web. Un ejemplo de estos sistemas es mongoDB[40]

- Orientadas a Objetos: almacenan la información representándola como objetos. Combinan conceptos de Programación orientada a objetos (OOP) con conceptos de bases de datos. Ofrecen características de OOP como encapsulación de datos, poliformismo o herencia. Se establece un claro paralelismo con los SGBDR donde una clase equivale a una tabla, un objeto a una fila y un atributo a una columna. Es especialmente útil cuando se trata con información compleja, estructuras de objetos cambiantes o por la naturaleza de los datos se pueden aprovechar conceptos de OOP aunque su escalabilidad es menor y están limitadas a un lenguaje de programación (por ejemplo, ObjectDB sólo puede ser utilizado con JAVA). Un ejemplo de estos sistemas es ObjectDB[41].

- Orientada a Grafos: almacenan la información en forma de grafos (existen SGBD que trabajan con grafos de manera nativa y otras que no, siendo más eficientes las primeras). Un grafo consiste en un conjunto de nodos y aristas donde los nodos actúan como objetos y las aristas como relaciones entre estos objetos. Tanto los nodos como las aristas pueden tener atributos. Sobresalen en entornos donde la complejidad de los datos es alta y estos están altamente conectados entre sí. Es altamente escalable y son los únicos sistemas que soportan ACID. Un ejemplo de estos sistemas es Neo4J[42].

Partiendo de las características de los datos (véase Sección 2.3). Se eliminan los SGBDR debido a sus limitaciones en cuanto a consulta de datos altamente relacionados y la rigidez de su esquema.

Interrogación de los datos: estos sistemas no están diseñados para preguntas complejas que impliquen un recorrido de caminos de varios niveles de profun-

didad, el motivo es la manera en que se almacenan físicamente las relaciones. Cuando las consultas dependen en gran medida de recorrer los datos a través de sus relaciones en un sistema relacional, se acaba con múltiples operaciones *join* que reducen la velocidad de procesamiento de consultas.

Esquema: la rigidez de su esquema no permite cambiarlo fácilmente ni realizar cambios en el modelo de una manera eficiente. El entendimiento de un dominio tan variable y cambiante asegura un cambio/actualización del modelo casi constante. Además, la heterogeneidad de los datos da como resultado que elementos del mismo tipo difieran en sus atributos.

De modo que sólo queda seleccionar un SGBD NoSQL. Como se ha visto anteriormente, la mayoría de estos sistemas trabajan con esquemas de datos simples (Clave-Valor, orientadas a documentos) y no garantizan ACID. Este factor limita a dos las posibles opciones: SGBD orientadas a objetos (recomendada para estructuras de datos complejas, no obstante, tampoco garantizan ACID) o a grafos. Nuestros datos están altamente conectados, su importancia reside en cómo se conectan entre ellos y tras estudiar el dominio, éste puede ser representado de manera natural como un grafo. Parece lógica utilizar la opción de un SGBD orientada a grafos, pero ¿qué dice la literatura? Como vemos en [43, 44, 45, 46, 47] el mundo de la bioinformática y la genética está haciendo uso de manera muy significativa de esta tecnología para abordar la gestión de sus datos, de gran complejidad como hemos visto anteriormente. Queda pues más que justificado el uso de un SGBD orientado a grafos para nuestra solución; más concretamente, nos hemos decantado por Neo4J ya que es el SGBD más maduro y muy por encima de sus competidores (En la Subsección 3.3.3 se explican en mayor profundidad las ventajas que aporta un SGBD orientado a grafos).

3.3.2. Teoría de Grafos

Se ha utilizado un SGBD orientado a grafos en el desarrollo de la herramienta. Es importante conocer las equivalencias terminológicas del ámbito de los grafos donde los vértices pueden ser llamados nodos y las aristas relaciones. Cabe preguntarse qué es exactamente un grafo y cómo se define formalmente. Partiendo de [48], un grafo es una representación de un conjunto de puntos y cómo estos puntos están conectados entre sí; adicionalmente, pueden añadirse al grafo aquellas propiedades que se consideren relevantes. La disciplina encargada de estudiar los grafos es la teoría de grafos A continuación se detallan algunos conceptos relevantes de la teoría de grafos.

Grafo simple Un grafo simple consiste en un conjunto finito no vacío de elementos llamados vértices (nodos) y un conjunto finito de parejas de elementos distintos llamados aristas (relaciones). Un grafo simple se caracteriza por tener como máximo una arista entre dos vértices.

Grafo Al contrario que con los grafos simples, un grafo general o simplemente grafo permite la existencia de múltiples aristas entre dos nodos y de bucles (aristas que relacionan un nodo consigo mismo).

Subgrafo Dado un grafo, un subgrafo es un subconjunto de dicho grafo. Todos los vértices y aristas de un subgafo están presentes en el grafo padre.

Adyacencia Aplicado a vértices, dos vértices son adyacentes si existe una arista que los una; del mismo modo, dos aristas son adyacentes si tienen un vértice en común. Dado un vértice, su grado se calcula como el número de aristas que comunican dicho vértice siendo un vértice de grado cero un vértice aislado

Camino Dado un grafo, un camino es una secuencia finita de aristas donde todos los pares de aristas consecutivas son adyacentes y distintas y todos los vértices del camino son distintos.

Grafo conectado Un grafo conectado es un grafo para el cual no es posible obtener dos subgrafos que no estén conectado entre sí, es decir, que no exista una arista entre un vértice del primer conjunto y un vértice del segundo. En otras palabras, un grafo conectado es aquel para el cual para todas las parejas de aristas existe un camino. En caso contrario, hablamos de un grafo no conectado.

Grafo dirigido En un grafo dirigido (dígrafo) las aristas que conforman dicho grafo son unidireccionales, es decir, solo pueden ser recorridos en una dirección y poseen un vértice origen y otro de fin.

Grafo de propiedades etiquetadas Es un tipo de grafo en el cual tanto los vértices como las aristas pueden tener asociados propiedades. Estas pueden almacenarse de diversas formas, por ejemplo, en pares de tipo clave-valor. Además, los vértices pueden clasificarse con etiquetas.

3.3.3. Características de los SGBD orientados a Grafos

Una vez ha sido presentado el concepto de grafo y los distintos tipos de grafos se puede proseguir con el estudio de los SGBD orientados a grafos (Subsección 3.3.2). Este tipo de bases de datos proporcionan un entorno que aprovecha y prima los datos altamente complejos, dinámicos e interconectados (información con un alto nivel de relaciones y asociaciones). La información conectada es aquel conjunto de datos para los cuales la correcta interpretación y extracción de valor requiere entender primero cómo se conectan y relacionan dichos datos.

Los SGBD orientados a grafos poseen una serie de particularidades que las hacen únicas. Por un lado, se trata de un SGBD que no sigue el modelo relacional; por otro lado, también se distancia del resto de familias que conforman el ecosistema NoSQL: son los únicos que mantienen ACID y se orientan a datos altamente interconectados (la tendencia general de los SGBD NoSQL es a ofrecer Disponibilidad Básica, *Soft-state*, consistencia eventual (BASE)[10] y almacenar datos siguiendo esquemas simples).

[10] el SGBD funcionará la mayoría del tiempo, el estado del sistema puede cambiar con el tiempo, aunque no reciba *input*, la integridad no se mantiene en todo momento, pero siempre se terminará alcanzando en algún momento (consistencia eventual)

A continuación, estudiamos las características que presentan los SGBD orientados a grafos a nivel general. Posteriormente se entrará en detalle en la implementación de dichas características por parte de Neo4J.

Basándonos en [49], exploraremos las características de las bases de datos orientadas a grafos. En primer lugar, a modo de resumen y, posteriormente, se profundizará en mayor detalle. Existen dos propiedades que deberían tenerse en cuenta:

- Almacenamiento interno de los datos: no todas las bases de datos orientadas a grafos tienen almacenamiento interno nativo de grafos. Un almacenamiento interno nativo de grafos permite almacenar y gestionar grafos de una manera mucho más eficiente.

- Motor de procesamiento: el procesamiento nativo de grafos consiste en el uso de la adyacencia libre de índices para mejorar el rendimiento de consultas que realizan recorridos de grafo.

Almacenamiento Nativo de Grafos

Como veremos a continuación, la técnica de adyacencia libre de índices es la clave para recorrer grafos de una manera eficiente, por lo tanto, es de vital importancia la forma en que estos grafos se almacenan. Los datos del grafo se almacenan en diferentes ficheros según el tipo de datos que sean; así, existen diferentes ficheros para nodos, relaciones, etiquetas, propiedades, transacciones, etc. Se consigue una división de responsabilidades entre los distintos ficheros (especialmente entre la estructura del grafo y los datos) que incrementa la velocidad a la que se recorre el grafo.

El almacenamiento de los datos está separado entre los elementos del grafo (nodos y relaciones) y la información de estos (atributos) con el objetivo de optimizar al máximo el recorrido de grafos. La información se almacena mediante registros de tamaño fijo; esto, unido al uso de identificadores de registro permite que recorrer el grafo se traduzca en buscar punteros en torno a una estructura de datos. Recorrer una relación de un nodo a otro se realiza mediante una operación con los identificadores.

Procesamiento Nativo de Grafos

El procesamiento nativo de grafos se realiza mediante el uso de adyacencia libre de índices. Un motor de procesamiento que utilice adyacencia libre de índices implica que cada nodo mantiene referencias directas (físicas) con sus nodos adyacentes. Por lo tanto, cada nodo actúa como un microíndice de sus nodos cercanos (esta aproximación es mucho más eficiente que utilizar índices globales). Como consecuencia, el tiempo de ejecución de las consultas es independiente del tamaño del grafo, siendo proporcional al tamaño del grafo buscado.

La Figura 3.6 ilustra la diferencia de eficiencia entre utilizar y no utilizar adyacencia libre de índices siendo x el número de pasos que hay que dar para atravesar un camino e y el coste temporal. La búsqueda mediante índices, dependiendo

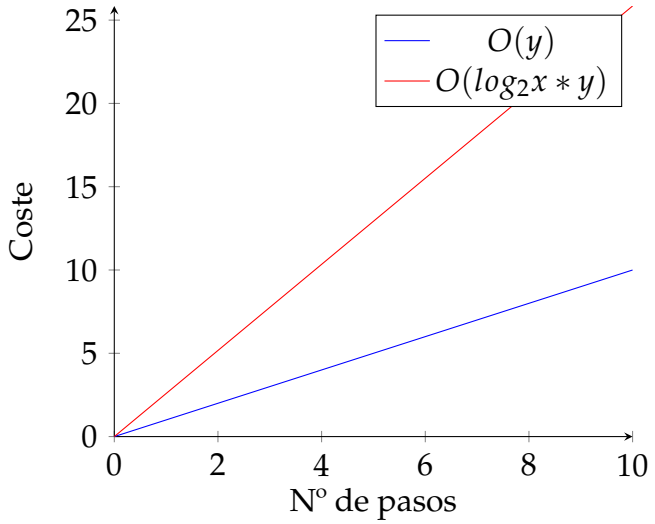

Figura 3.6: Comparativa de uso de adyacencia libre de índices

de la implementación, puede llegar a ser de coste $O(log_2 x)$ mientras que utilizando adyacencia libre de índices el coste es de $O(1)$ para relaciones inmediatas. Por tanto, cuando se atraviesa un camino de y pasos, los costes pasan a ser $O(y * log_2 x)$ y $O(y)$ respectivamente. Esta característica permite que las operaciones *join* se precomputen y almacenen en la base de datos como relaciones. Las relaciones pueden ser recorridas muy eficientemente y de manera bidireccional.

Neo4J

Una vez analizados los conceptos generales de las bases de datos orientadas a grafos, pasamos a estudiar cómo son implementadas en Neo4J. Para ello, nos hemos basado en [49]. Exploraremos las características de Neo4J cuyo modelo implementa *labeled property graph*. Como se ha mencionado anteriormente, existen dos propiedades importantes a la hora de tratar con bases de datos orientadas a grafos:

- Almacenamiento interno de los datos: Neo4J tiene almacenamiento nativo de grafos, estando optimizado para almacenar y gestionar grafos (no todas las bases de datos orientadas a grafos tienen almacenamiento nativo de grafos).
- Motor de procesamiento: el motor de procesamiento de Neo4J hace uso de la adyacencia libre de índices para mejorar el rendimiento de consultas que realizan recorridos en los grafos.

Neo4J posee procesamiento nativo de grafos debido a que hace uso de adyacencia libre de índices, es decir, cada nodo mantiene referencias directas (físicas) con sus nodos adyacentes y actúa como un microíndice de sus nodos cercanos. La técnica de adyacencia libre de índices es clave para recorrer grafos de una manera eficiente. Partiendo de la arquitectura de alto nivel de Neo4J (Figura 3.7) se ha analizado cómo este sistema almacena de manera física la información existente. Los datos del grafo están separados en diferentes ficheros con el objetivo de optimizar al máximo el recorrido del grafo.

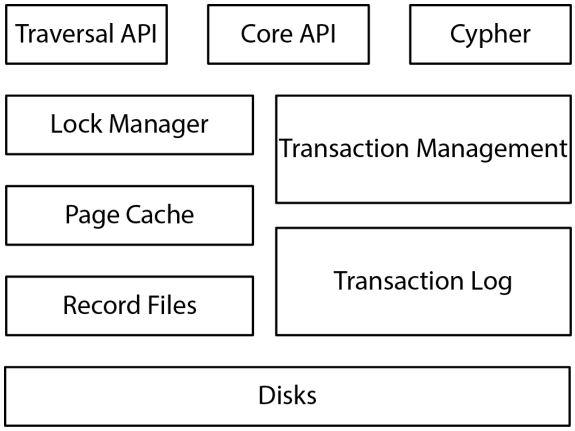

Figura 3.7: Arquitectura de Neo4J

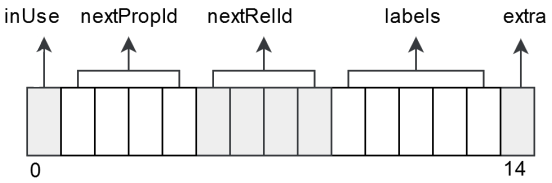

Figura 3.8: Estructura física de un nodo

Los nodos se almacenan de manera física en registros de 15 bytes en el fichero de almacenamiento de nodos, llamado *neostore.nodestore.db* y su estructura puede verse en la Figura 3.8. El hecho de tener una estructura de tamaño fija permite buscar nodos de manera rápida ya que, por ejemplo, un nodo cuyo id sea 100 se sabe que empieza a 1500 bytes desde la posición de inicio del fichero, lo que el sistema puede buscar nodos con coste O(1). El primer byte es un *flag* indica si el registro está en uso en ese momento. Los siguientes cuatro bytes representan el identificador de la primera relación conectada con dicho nodo. Los siguientes cuatro bytes indican el identificador del primer atributo del nodo. Los siguientes cinco bytes se utilizan para referencias las etiquetas de dicho nodo. Finalmente, queda un byte reservado para *flags* y futuros usos. Un registro de un nodo es simple y ligero, tan sólo contiene punteros a listas de relaciones, propiedades y etiquetas.

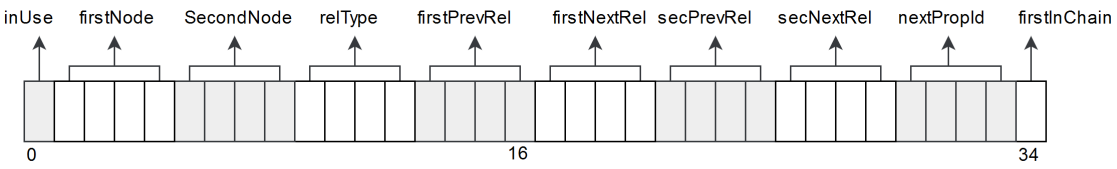

Figura 3.9: Estructura física de una relación

Las relaciones se almacenan de manera física de 35 bytes en el fichero de almacenamiento de relaciones, llamado *neostore.relationshipstore.db* y su estructura puede verse en la Figura 3.9. El primer byte, al igual que con los nodos, indica si el registro está en uso de ese momento. A continuación, se utilizan ocho bytes para indicar los nodos de las relaciones, cuatro para el nodo de inicio y cuatro para el nodo de fin. Los siguientes cuatro bytes identifican el tipo de relación. Los siguientes dieciséis bytes contienen referencias a la siguiente y anterior relación

de cada uno de los nodos. Por último, queda un byte para indicar si el registro es el inicio de una *cadena de relaciones*.

El almacenamiento de los datos está separado entre los elementos del grafo (nodos y relaciones) y la información de estos (atributos) con el objetivo de optimizar al máximo el recorrido de grafos. Esta estructura puede verse mejor con un ejemplo de cómo se almacenaría físicamente un pequeño grafo (Figura 3.10).

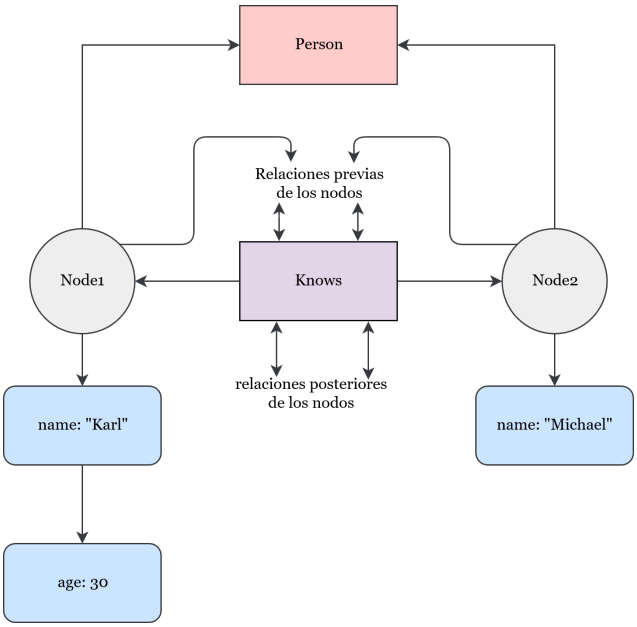

Figura 3.10: Grafo almacenado físicamente

En la Figura 3.10 se observa cómo hay dos nodos presentes. Ambos nodos tienen contienen un puntero a su primer atributo (`name`, su primera etiqueta (`Person`) y su primera relación para las que es nodo de inicio o fin. Dicha relación se encuentra dentro de una lista doblemente enlazada. Las relaciones se deben entender como una entidad que "pertenece" a los nodos de dicha relación. Cada relación contiene punteros a: el primer atributo de la relación (si lo tuviera), la relación predecesora y sucesora el nodo de inicio y la relación predecesora y sucesora del nodo de fin. En nuestro ejemplo, la relación `Knows` no es su primera relación, sino que se ha accedido recorriendo la lista doblemente enlazada de relaciones.

Como hemos explicado anteriormente, el recorrido de un grafo consiste en buscar y realizar operaciones sobre punteros de datos. Por ejemplo, basándonos en el esquema físico implementado por Neo4J, para recorrer un camino se ejecutan los siguientes pasos:

1. Dado un nodo, se obtiene el ID de su relación (tercer campo) y se calcula la posición en que se encontrará el registro de la relación multiplicando dicho número por el tamaño de registro de relaciones (35 bytes). De este modo, se llega directamente al registro deseado.

2. Desde la relación, obtener el ID del segundo nodo (campo 3) y multiplicar dicho valor por el tamaño de registro de nodos (15 bytes)

Con dos simples multiplicaciones hemos realizado el recorrido de una relación. Este procedimiento es igualmente aplicable cuando se desea filtrar por tipo de relación o por etiqueta, únicamente hay que realizar las multiplicaciones para obtener los registros deseados y comprobar si cumplen las condiciones.

Por otra parte, separados físicamente en ficheros distintos de la estructura del grafo, se encuentran los ficheros de atributos (o propiedades, entiéndanse como sinónimos), que almacenan datos de tipo clave-valor. Estos atributos pueden ser referenciados tanto por los nodos como por las relaciones. Al igual que con los nodos y las relaciones, poseen una longitud fija y se almacenan en el fichero *neostore.propertystore.db*. Cada registro de atributo está compuesto de cuatro bloques de propiedades y un puntero a la siguiente propiedad; cada atributo puede ocupar hasta cuatro bloques por lo que cada registro contendrá, como máximo, cuatro atributos. De cada atributo se almacena su tipo y un puntero al fichero *neostore.propertystore.db.index* que almacena el nombre del atributo. En cuanto al valor del atributo, pueden darse dos opciones: que el registro contenga el valor o que contenga un puntero a un registro dinámico. Estos registros se utilizan para almacenar propiedades de un tamaño grande. Existen dos ficheros para estos registros: *neostore.propertystore.db.string* para almacenar cadenas de texto y *neostore.propertystore.db.array* para almacenar arrays de datos. Estos registros dinámicos son de tamaño fijo y se estructuran como una lista enlazada, de modo que un atributo muy largo puede ocupar más de un registro dinámico.

3.3.4. Proceso ETL

Todo el proceso ETL se ha basado en el ECGCit.. Gracias a este hemos podido establecer una relación directa con los diferentes ficheros. Todo el proceso de transformación se ha realizado sobre el conocimiento obtenido en el proceso de creación del ECGCit.. Ha sido, por un lado, una guía omnipresente en todo el proceso y, por otro lado, el mecanismo de validación del mismo.

En esta subsección se explica el proceso de transformación de datos de los formatos originales a los ficheros Fichero de valores separados por coma, del inglés *comma-separated values* (CSV) utilizados para la generación de la base de datos. Neo4J permite cargar datos de dos maneras: mediante la instrucción `LOAD CSV` accediendo a la base de datos vía web o con la herramienta `neo4j-admin import`. La primera está pensada para conjuntos de datos medianos mientras que el segundo está pensado para cantidades de datos mucho mayores. Debido al volumen de nuestros datos, se ha optado por la segunda opción.

Esta herramienta toma como datos de entrada un conjunto de ficheros en formato CSV: uno por cada nodo y uno por cada relación. Estos ficheros requieren una cabecera para identificar los campos la cual puede encontrarse dentro de los ficheros o en un fichero externo. La cabecera debe seguir un formato establecido de modo que, para el caso de los nodos, cada atributo se compondrá de su nombre y, opcionalmente el tipo del atributo separado por dos puntos (si no se especifica se asumirá que es una cadena de texto). Uno de estos atributos debe ser el identificador universal del nodo. Este atributo se identificará mediante la palabra ID separada por dos puntos del atributo (opcionalmente se puede especificar un espacio de identificador de modo que no sean identificadores globales).

La última columna debe contener las etiquetas correspondientes al nodo, si hay más de una etiqueta estas deben estar separadas por coma. Para las relaciones, la primera columna contiene el identificador del nodo que inicia la relación, los atributos de la relación (si los tiene), el identificador del nodo que finaliza la relación y el tipo de relación (del mismo modo que las etiquetas en los nodos).

En la Figura 3.11 se observa el proceso de transformación de datos esquematizado en cinco pasos:

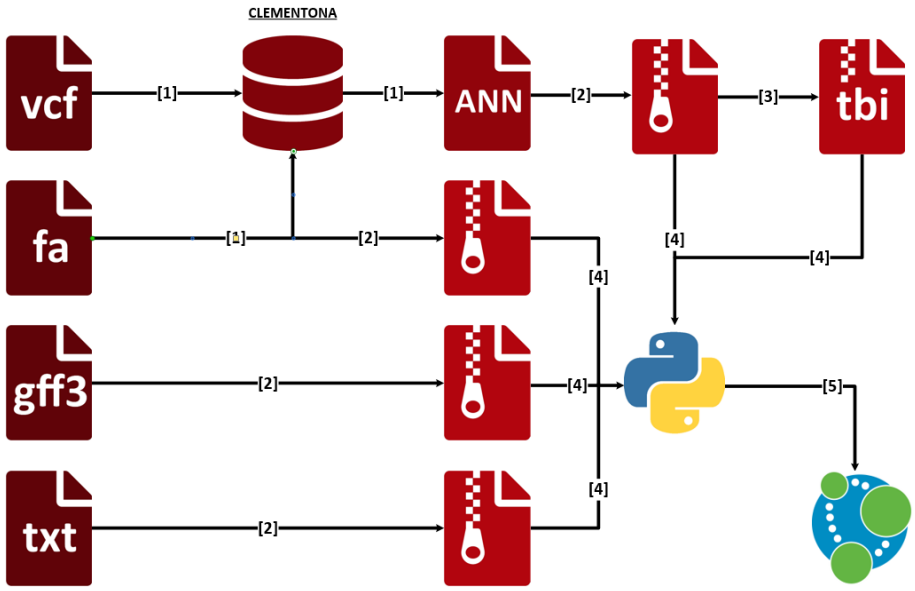

Figura 3.11: Proceso ETL esquematizado

1. Obtención de los ficheros anotados mediante la herramienta SnpEff (necesaria la creación de una base de datos con ficheros fasta). Este paso ya ha sido explicado en la sección anterior (Sección 3.1).

2. Compresión de los ficheros a formato .gz con la herramienta Bgzip.

3. Indexado de los ficheros VCF anotados con la herramienta tabix.

4. Ejecución de scripts en Python con el objetivo de obtener los ficheros CSV para la creación de la base de datos.

5. Creación de la base de datos mediante la herramienta `neo4j-admin import`.

Para aumentar la modularidad a la hora de creación de la base de datos se ha decidido crear múltiples scripts, de modo que cada uno genere (en la medida de lo posible teniendo en cuenta el impacto temporal de la ejecución de los scripts) un único tipo de nodo o un único tipo de relación. Con el objetivo de reducir al máximo los tiempos se han utilizado dos herramientas extra:

Bgzip[11]

Se trata de una herramienta de compresión con una interfaz similar a la herramienta gzip[12] que utiliza múltiples técnicas de compresión (*Burrows-Wheeler transform* (BWT) entre otras) para comprimir archivos con un alto índice de compresión. Esta herramienta se ha utilizado para reducir el tamaño de todos los ficheros VCF. Al comprimir los ficheros VCF anotados obtenemos de media una reducción del tamaño del noventa por cien para lo cual se ha utilizado el comando referenciado en el Fragmento de código A.3 . El tamaño de los ficheros se reduce en casi un 90 %: los ficheros con extensión .gff3 pasan de 100 MB a 9.2 MB, los ficheros de texto (con extensión .txt) pasan de 12 MB a 1.4MB, los ficheros VCF anotados pasan de 90 GB a 8.9 GB y los ficheros con extensión .fa pasan de 309 MB a 90 MB.

Tabix[13]

Esta herramienta indexa ficheros VCF. Recibe como entrada un fichero comprimido en formato Bgzip y crea un fichero auxiliar que actúa como índice (fichero con formato .tbi). Tras este proceso de indexación, Tabix puede recuperar rápidamente las lineas de datos que se superponen a las regiones específicadas en el formato "chr: beginPos-endPos". En otras palabras, crea un diccionario cuya clave es el scaffold y el valor la lista de variaciones dentro de dicho scaffold. Se han indexado todos los ficheros VCF comprimidos mediante el comando del Fragmento de código A.4

Para el resto de ficheros no se ha realizado ninguna transformación o preprocesado antes de la ejecución de los scripts principales a parte de su compresión. El proceso de transformación completo para los ficheros VCF sería:

1. Adición de las anotaciones con la herramienta SnpEff.
2. Compresión de los ficheros anotados con la herramienta Bgzip.
3. Indexado de los ficheros comprimidos con la herramienta Tabix.

Python

Los scripts han sido generados en Python debido a las facilidades que ofrece en la manipulación de cadenas de texto y porque, al ser utilizado en el ámbito bioinformático, dispone de diversos paquetes que nos ayudan al procesado de los ficheros.

Para la correcta ejecución de los scripts es necesario instalar las siguientes dependencias:

– PyVCF[14]: módulo encargado de procesar ficheros VCF utilizando los metadatos de los ficheros.

[11] https://sourceforge.net/p/bgzip/wiki/Home/
[12] https://www.gnu.org/software/gzip/
[13] http://www.htslib.org/doc/tabix.html
[14] https://github.com/jamescasbon/PyVCF

- Pysam[15]: módulo que facilita la lectura y manipulación de secuencias de datos en ficheros SAM/BAM. Actúa como *wrapper* de la librería en C HTSlib.
- Cython[16]: superconjunto del lenguaje Python con soporte para llamadas a funciones en C y declaración de tipos de C. Permite combinar Python y C gracias a su compilador estático que genera código C a partir de código en Cython. Ampliamente utilizado como *wrapper* de librerías externas de C.

En el proyecto se han dividido los scripts en función de si generan nodos, aristas o ambos. Empezando por los scripts que generan únicamente nodos existen cinco scripts. Todos los scripts tienen un conjunto de métodos comunes:

- Inicialización de la clase creada: se asocian los ficheros de entrada y salida, se crean las estructuras de datos necesarias y se configura el *logger* que almacenará la información necesaria.
- Método principal: método encargado de la ejecución del script, en la mayoría de los casos consta de tres pasos: la carga de los datos, el procesado de los datos y la escritura de los datos procesados.
- Método de carga: realiza la carga y procesado de los datos presentes en los ficheros de entrada.
- Método de escritura: escritura de los datos procesados en los ficheros CSV correspondientes.

Se han creado cinco scripts que generan únicamente nodos, sin relaciones. El primero de ellos se encarga de crear los nodos correspondientes a dominios (Tabla 3.27).

domain.py

Entrada	**Salida**
Cclementia_v1.0_genes2IPR.txt.gz	domains.csv
Cabecera del fichero	
(1)`id:ID(DOMAIN-ID);description;:LABEL`	

Tabla 3.27: Datos del script domain.py

Por cada línea genera una nueva entrada con su identificador y descripción, como se puede ver en la Figura 3.12 evitando incluir aquellos valores duplicados. En total, se generan 5813 nodos (0.3 MB).

```
map00312;ec:3.5.2.6;intervenes
```

Figura 3.12: Ejemplo fichero generado con pathway_enzyme.py

[15] http://pysam.readthedocs.io/en/latest/api.html
[16] http://cython.org/

El segundo script crea los nodos de elementos cromosómicos, seis en total (gen, mRNA, exón, UTR 5', CDS, UTR 3') (Tabla 3.28).

elements.py

Entrada	Salida
Cclementina_182_v1.0.gene_exons.gff3.gz Cclementina_v1.0_scaffolds.fa.gz	genes.csv (1) mrna.csv (2) exon.csv (3) five.csv (4) cds.csv (5) three.csv (6)

Cabecera de los ficheros

(1) `scaffold:int;start:int;sequence;end:int;direction;id:ID(ELEMENT-ID);name;:LABEL`
(2) `scaffold:int;start:int;sequence;end:int;direction;id:ID(ELEMENT-ID);pacid:int;longest:int;name;:LABEL`
(3), (4), (5) y (6) `scaffold:int;start:int;sequence;end:int;direction;id:ID(ELEMENT-ID);pacid:int;:LABEL`

Tabla 3.28: Datos del script elements.py

La primera acción de este script es optimizar la obtención de subcadenas de aminoácidos pertenecientes a los scaffolds. Para ello, ejecuta el método `optimize_sequence` que, a partir del fichero de secuencias de scaffolds (`Cclementina_v1.0_scaffolds.fa.gz`) crea un diccionario cuyas entradas tiene como clave el identificador del scaffold y como valor su secuencia de aminoácidos completa (Fragmento de código A.24). Una vez realizada la optimización se procede al procesamiento de los datos. Por cada fila comprueba el elemento del que se trata y ejecuta la función de dicho tipo de elemento, que extrae los campos necesarios de la línea procesada (véase el apartado "Cabecera de los ficheros" de la Tabla 3.28) y la cadena que lo compone (utilizando el diccionario de scaffolds). En total se han generado 412.967 nodos (247 MB de datos). A continuación, en la Figura 3.13, puede verse un ejemplo de los datos escritos por el script `elements.py`.

El tercer script se encarga de la generación de los nodos de las enzimas (Tabla 3.29).

enzyme.py

Entrada	Salida
kegg_txt_20110915_1636.txt.gz Cclementina_v1.0_KEGG-orthologs.txt.gz	enzymes.csv

Cabecera del fichero

`id:ID(ENCIMA-ID);n_seq:int;description;:LABEL`

Tabla 3.29: Datos del script enzyme.py

(1)
`1;23527053;AAC...TAT;23542559;+;Ciclev10007219m.g.v1.0;Ciclev10007219m.g;gene`
(2)
`1;23527053;GTTCAGCTA...CTATAGCTAA;23542559;+;Ciclev10007219m.v1.0;20793483;1;Ciclev10007219m;mrna`
(3)
`1;23527053;AATA...TGAATC;23527358;+;Ciclev10007219m.v1.0.exon.1;20793483;exon`
(4)
`1;23527053;ACAGCTT...CATAGCG;23527325;+;Ciclev10007219m.v1.0.five_prime_UTR.1;20793483;five_prime_UTR`
(5)
`1;23527326;ACAGCA...AGTGAT;23527358;+;Ciclev10007219m.v1.0.CDS.1;20793483;cds`
(6)
`1;23541959;ACACTTG...GATGT;23542023;+;Ciclev10007219m.v1.0.three_prime_UTR.1;20793483;three_prime_UTR`

Figura 3.13: Ejemplo ficheros generados con elements.py

Las enzimas pueden encontrarse en dos ficheros de modo que el proceso de carga y procesado de datos está dividido en dos subprocesos.

- `kegg_txt_20110915_1636.txt.gz`: se disponen de todos los campos por lo que tan sólo hay que extraerlos y almacenarlos.
- `Cclementina_v1.0_KEGG-orthologs.txt.gz`: únicamente se dispone del identificador, por lo tanto los demás campos no se crean. En cada línea puede aparecer más de una enzima y estas pueden no ser enzimas concretas sino grupos más genéricos, en cuyo caso se ignoran. Se obtiene el campo correspondiente y se almacena.

Una vez se han obtenido se eliminan duplicados de ambos ficheros y se vuelcan en los ficheros CSV correspondiente. En total se han generado 1.273 nodos (0.05 MB de datos). A continuación, en la Figura 3.14, puede verse un ejemplo de los datos escritos por el script `enzyme.py`.

`ec:3.5.2.6;3;beta-lactamase;enzyme`

Figura 3.14: Ejemplo fichero generado con enzyme.py

El cuarto script crea los nodos de los grupos ortólogos (Tabla 3.30).

orthologs.py

Entrada	**Salida**
Cclementina_v1.0_KEGG-orthologs.txt.gz	orthologs.csv
Cabecera del fichero	
`id:ID(ORTHOLOG-ID);description;:LABEL`	

Tabla 3.30: Datos del script orthologs.py

Por cada fila del fichero se obtiene un grupo ortólogo eliminando aquellos que estén duplicados. En total se han generado 2.271 nodos (0.1 MB de datos). A continuación, en la Figura 3.15, puede verse un ejemplo de los datos escritos por el script `orthologs.py`.

```
K01810;glucose-6-phosphate isomerase [EC:5.3.1.9];ortholog
```

Figura 3.15: Ejemplo fichero generado con orthologs.py

El quinto script crea los nodos de los pathways (Tabla 3.31).

pathways.py

Entrada	**Salida**
Cclementina_v1.0_KEGG-pathways.txt.gz kegg_txt_20110915_1636.txt.gz	pathways.csv
Cabecera del fichero	
`id:ID(ORTHOLOG-ID);description;:LABEL`	

Tabla 3.31: Datos del script pathways.py

El caso de los pathways es un tanto especial ya que al provenir de dos archivos distintos pueden tener identificadores distintos. Por ello, se ha utilizado la descripción del pathway, que sí coincide, para unificar los identificadores y utilizar como identificador único y final la concatenación de ambos. Por cada fila del fichero se obtiene un grupo ortólogo evitando duplicados. En total se han generado 330 nodos (0.02 MB de datos). A continuación, en la Figura 3.16, puede verse un ejemplo de los datos escritos por el script `pathways.py`.

```
K01810;glucose-6-phosphate isomerase [EC:5.3.1.9];ortholog
```

Figura 3.16: Ejemplo fichero generado con pathways.py

Se han creado tres scripts que generan tanto nodos como relaciones. El primero de ellos se encarga de crear los nodos y relaciones correspondientes a GO (Tabla 3.32).

go.py

Entrada	**Salida**
Cclementia_v1.0_genes2GO.txt.gz Cclementina_182_v1.0.gene_exons.gff3.gz	go.csv (1) mrna_go.csv (2) gene_go.csv (3)
Cabeceras de los ficheros	
(1)`id:ID(go);type;description;:LABEL` (2) y (3) `:START_ID(ELEMENT-ID);:END_ID(go);:TYPE`	

Tabla 3.32: Datos del script go.py

Por cada línea genera una nueva entrada con su identificador, tipo y descripción (evitando nodos duplicados) y la relación del nodo con los elementos gen y mRNA (cuyos identificadores se obtienen mediante el método `get_mrna_id` referenciado en el Fragmento de código A.27). En total, se generan 1667 nodos, y

119.413 relaciones (5.2 MB). A continuación, en la Figura 3.17, puede verse un ejemplo de los datos escritos por el script go.py.

(1) `0005488;Molecular Function;binding;GO`
(2) `Ciclev10000001m.v1.0;0005488;associated`
(3) `Ciclev10000001m.g.v1.0;0005488;associated`

Figura 3.17: Ejemplo fichero generado con go.py

El segundo script crea los nodos de proteínas y sus relaciones con los elementos gen y mRNA (Tabla 3.33).

proteins.py

Entrada	**Salida**
Cclementina_182_v1.0.protein.fa.gz	proteins.csv (1) protein_mrna.csv (2) protein_gene.csv (3)
Cabeceras de los ficheros	
(1)`id:ID(go);type;description;:LABEL` (2) y (3) `:START_ID(ELEMENT-ID);:END_ID(go);:TYPE`	

Tabla 3.33: Datos del script proteins.py

El proceso de carga recorre el fichero y procesa los datos. Por cada línea almacena los nodos de tipo proteína con atributos necesarios (sigue una estructura donde la información de la proteína está en una línea cuyo carácter inicial es ">" y, hasta la siguiente línea del mismo tipo, el resto de líneas son la secuencia de la proteína) y las relaciones con el gen y el mRNA (identificadores presentes en el fichero). En total se generan 33.929 nodos y 67.858 relaciones (19 MB de datos). A continuación, en la Figura 3.18, puede verse un ejemplo de los datos escritos por el script proteins.py.

(1) `Ciclev10013962m;MCSYLLIGRPIAA...EGPTPISAATMVAAGIFL;protein`
(2) `Ciclev10013962m.v1.0;Ciclev10013963m;mrna_encodes_protein`
(3) `Ciclev10013962m.g.v1.0;Ciclev10013963m;gene_encodes_protein`

Figura 3.18: Ejemplo fichero generado con proteins.py

El tercer script procesa los ficheros VCF anotados. Se trata de uno de los scrips más costosos y que más nodos y relaciones genera. En cuanto a nodos, genera: Variation, Lecture, Annotation, NMD y LOF; en cuanto a relaciones, genera relaciones entre: Variation y Lecture, Annotation y Lecture, NMD y Lecture, LOF y Lecture, Annotation y Gene, NMD y Gen y LOF y Gen.

variations.py

Entrada	Salida
prueba_\w_(indel|snp)\.vcf\.gz[17]	variations.vcf (1) lectures.csv (2) variations_lectures.csv (3) annotations.csv (4) nmd.csv (5) lof.csv (6) ann_var.csv (7) nmd_var.csv (8) lof_var.csv (9) ann_gen.csv (10) ann_mrna.csv (11) lof_gen.csv (12) nmd_gen.csv (13)

Cabeceras de los ficheros

(1)`id:ID(VARIATION-ID);type;scaffold:int;POS:int;REF;ALT;:LABEL`

(2):`ID(LECTURE-ID);Variety;scaffold;QUAL:float;FILTER;GT;AD:float[];DP:int;GQ:int;PL:float[];AC:float[];AF:float[];AN:int;BaseQRankSum:float;FS:float;HaplotypeScore:float;InbreedingCoeff;MLEAC:float[];MLEAF:float[];MQ:float;MQ0:int;MQRankSum:float;QD:float;RPA:int[];RU;ReadPosRankSum:float;STR;:LABEL`

(3):`START_ID(LECTURE-ID);:END_ID(VARIATION-ID);:TYPE`

(4):`ID(ANNOTATION-ID);allele;annotation:string[];annotation_impact;feature_type;transcript_biotype;rank;hgvs.c;hgvs.p;cdna.pos/length;cds.pos/length;aa.pos/length;distance:int;errors/warnings/infos;intergenic:boolean;:LABEL`

(5):`ID(NMD-ID);num_transcripts_gene:int;perceent_transcript_affected:int;:LABEL`

(6):`ID(LOF-ID);num_transcripts_gene:int;percent_transcript_affected:int;:LABEL`

(7):`START_ID(ANNOTATION-ID);:END_ID(VARIATION-ID);:TYPE`

(8):`START_ID(NMD-ID);:END_ID(VARIATION-ID);:TYPE`

(9):`START_ID(LOF-ID);:END_ID(VARIATION-ID);:TYPE`

(10) y (11):`START_ID(ANNOTATION-ID);order:int;:END_ID(ELEMENT-ID);:TYPE`

(12):`START_ID(LOF-ID);:END_ID(ELEMENT-ID);:TYPE`

(13):`START_ID(NMD-ID);:END_ID(ELEMENT-ID);:TYPE`

Tabla 3.34: Datos del script variations.py

Debido al volumen de datos procesados ha sido necesario utilizar multiproceso para reducir los tiempos. En el método principal se crean los objetos compartidos por los procesos y se inicia el procesamiento de los ficheros (véase Fragmento de código A.25). El primer proceso en crearse ejecuta el método `io_loop`. Este método ejecuta un bucle que comprueba si hay fragmentos de texto para escribir en ficheros. Si hay datos, bloquea la cola, escribe los datos y la libera; en caso contrario espera y vuelve a comprobar si hay datos para escribir. Cuando se han procesado todos los datos se realiza una última iteración para escribir

[17] expresión regular debido al largo gran número de ficheros. Los valores del campo \w pueden encontrarse en la figura 3.5

los registros faltantes que aún permanezcan en la cola y cierra los ficheros. En el Fragmento de código A.26 se muestra el código de dicho método.

En la línea 15 del Fragmento de código A.26 se llama al método `write_chunks`. Este método escribe los datos recibidos en los ficheros correspondientes. Una vez se ha lanzado este primer proceso, se inicia al tratamiento de los ficheros VCF ejecutando el método `load_and_write` sobre cada uno de estos. Los módulos `pysam` y `pyVCF` así como la herramienta tabix son utilizadas a partir de este momento. Esta función crea un *reader* del módulo `pyVCF` para los ficheros de modo que facilite la extracción de datos de las variaciones, las ordena por su cromosoma y genera, por cada cromosoma, un nuevo proceso (hasta un máximo de treinta y dos) que se encarga de procesar las variaciones de dicho cromosoma mediante el método `load_records`. Este método realiza los siguientes pasos:

- Obtención de las variaciones a procesar mediante el método `fetch` del módulo `pyVCF`. Este método obtiene, de manera óptima gracias al uso de ficheros indexados mediante la herramienta tabix todas las variaciones de un cromosoma dado en un fichero.
- Procesado de cada una de las variaciones. En este punto se generan las líneas que corresponden a los nodos y relaciones que se crearán en la base de datos. En este paso se ejecutan los siguientes métodos: `get_id` (genera el identificador de la variación, utilizado también para generar los identificadores de las anotaciones), `registros_variacion` (genera los nodos de tipo variación), `registros_lectura` (genera los nodos de tipo lectura), `get_annotations` (genera los nodos de tipo ANN y sus relaciones con variación, gen y mRNA), `get_lof` (genera los nodos de tipo LOF y sus relaciones con variación y gen) y `get_nmd` (genera los nodos de tipo NMD y sus relaciones con variación y gen)
- Adición de los datos almacenados en la cola compartida. Cuando se llega a un tamaño arbitrario (256 en nuestro caso) de variaciones procesadas, estas se añaden a la cola compartida para que las escriba el proceso principal. Esto permite que la memoria que ocupan los procesos no crezca más de lo debido.

En la Figura 3.19, puede verse un ejemplo de los datos escritos por el script `variation_elements.py`. Se obtienen 656.697.943 nodos y 1.247.695.385 relaciones (147 GB de datos).

(1) indel-15-14906-ATT-[A];indel;15;14906;ATT;[A];variation
(2) 119;000;15;273.73;PASS;0/1;27,9;41;99;311,0,966;1;0.5;2;2.484;13.382;
106.9558;;1;0.5;43.48;0;-3.361;6.68;10,8;T;-2.866;True;lecture
(3) 119;inde-15-14906-ATT-[A];lecture_from
(4) indel-15-14906-ATT-[A]57ddddfd8c;A;intergenic_region;
MODIFIER;;;;;;;;;;false;annotation
(5) indel-4-781502-G-[GAGGTA]dcb76767;1;100;nmd
(6) indel-4-781502-G-[GAGGTA]20dcb76767;1;100;nmd
(7) indel-15-14906-ATT-[A]57ddddfd8c;indel-15-14906-ATT-[A];add
(8) indel-4-781502-G-[GAGGTA]dcb76767;indel-4-781502-G-[GAGGTA;nmd]
(9) indel-108-7554-C-[CA]d41615356e;indel-108-7554-C-[CA];lof
(10) y (11) indel-26-29021-T-[TC]1452851a80;1;Ciclev10004101m.g.v1.0;
annotate_gene
(12) indel-4-781502-G-[GAGGTA]20dcb76767;Ciclev10031765m.g.v1.0;nmd_gene
(13) indel-108-7554-C-[CA]d51615356e;Ciclev10013959m.g.v1.0;lof_gene

Figura 3.19: Ejemplo ficheros generados con variations.py

En cuanto a los scripts que crean únicamente relaciones entre nodos, son seis. El primero de ellos se encarga de la generación de las relaciones entre los nodos Domain y Gene (Tabla 3.35).

domain_gene.py

Entrada	**Salida**
Cclementia_v1.0_genes2IPR.txt.gz Cclementina_182_v1.0.gene_exons.gff3.gz	domain_gene.csv
Cabecera del fichero	
:START_ID(ELEMENT-ID);:END_ID(DOMAIN-ID);:TYPE	

Tabla 3.35: Datos del script domain_gene.py

En el fichero de entrada se relacionan los dominios con mRNAs, no con genes. Por ello se ha creado un diccionario de datos que relaciona el nombre del mRNA con el identificador del gen al que pertenece (Fragmento de código A.27). De este modo, se relaciona cada dominio con el gen y se agrega como tipo de relación "associated". Se obtienen 74.486 relaciones (3.2 MB de datos). A continuación, en la Figura 3.20, puede verse un ejemplo de los datos escritos por el script `domain_gene.csv`.

Ciclev10000001m.g.v1.0;IPR000449;associated

Figura 3.20: Ejemplo fichero generado con domain_gene.py

El segundo script genera las relaciones entre los distintos elementos cromosómicos (Tabla 3.36).

elements.py

Entrada	Salida
Cclementina_182_v1.0.gene_exons.gff3.gz	elements_rel.csv (1) elements_base_rel.csv (2) elements_base_gen_rel.csv (3)
Cabeceras de los ficheros	
(1) y (2) `:START_ID(ELEMENT-ID);:END_ID(ELEMENT-ID);:TYPE` (3) `:START_ID(ELEMENT-ID);number:int;:END_ID(ELEMENT-ID);:TYPE`	

Tabla 3.36: Datos del script elements.py

entre gen y mRNA, exón. CDS, UTR 3' y UTR 5'; entre mRNA y exón. CDS, UTR 3' y UTR 5'; entre exón y CDS, UTR 3' y UTR 5'. El script recorre las líneas del fichero almacenando el gen, mRNA y el exón y genera las relaciones correspondientes según el tipo del elemento de la línea procesada. Se obtienen 1.177.451 relaciones (72 MB de datos). A continuación, en la Figura 3.21, puede verse un ejemplo de los datos escritos por el script `elements.py`.

(1) `Ciclev10007313m.v1.0;Ciclev10007313m.g.v1.0;mrna_gen`
(2) `Ciclev10007219m.v1.0.exon.9;Ciclev10007219m.v1.0;exon_mrna`
(3) `Ciclev10007225m.v1.0.CDS.16;16;Ciclev10007225m.g.v1.0;cds_gen`

Figura 3.21: Ejemplo ficheros generados con elements.py

El tercer script genera las relaciones entre las enzimas y los mRNAs que las codifican y los genes a los que pertenecen (Tabla 3.37).

enzyme_relations.py

Entrada	Salida
kegg_txt_20110915_1636.txt.gz Cclementina_182_v1.0.gene_exons.gff3.gz	enzyme_mrna.csv (1) enzyme_gene.csv (2)
Cabeceras de los ficheros	
`:START_ID(ELEMENT-ID);:END_ID(ENCIMA-ID);:TYPE`	

Tabla 3.37: Datos del script enzyme_relations.py

El fichero que contiene dicha relación no utiliza los identificadores del mRNA, sino su nombre, por lo tanto, al igual que con el script `domain_gene.py`, ha sido necesario ejecutar el método `get_mrna_id` para obtener sus identificadores y generar las relaciones correctamente. A continuación, en la Figura 3.22, puede verse un ejemplo de los datos escritos en el fichero `enzyme_relations.py`. Se obtienen 12.146 relaciones (0.6 MB de datos).

(1) `Ciclev10031473m.v1.0;ec:3.5.2.6;mrna_encodes_enzyme`
(2) `Ciclev10031473m.g.v1.0;ec:3.5.2.6;gene_encodes_enzyme`

Figura 3.22: Ejemplo ficheros generados con enzyme_relations.py

El cuarto script genera las relaciones entre los grupos ortólogos y, por un lado, las enzimas y, por otro, los genes (Tabla 3.38).

ortholog_relations.py

Entrada	**Salida**
Cclementina_v1.0_KEGG-orthologs.txt.gz	enzyme_orth.csv (1)
Cclementina_182_v1.0.gene_exons.gff3.gz	gene_orth.csv (2)
Cabeceras de los ficheros	
(1):`START_ID(ENCIMA-ID);:END_ID(ORTHOLOG-ID);:TYPE`	
(2):`START_ID(ELEMENT-ID);:END_ID(ORTHOLOG-ID);:TYPE`	

Tabla 3.38: Datos del script ortholog_relations.py

Se ha creado un método para obtener el identificador del gen, ya que en el fichero aparece con el nombre del mRNA no con el identificador del gen. Cada mRNA alacena en el campo "Parent" el gen al que pertenece. Para este fichero han tenido que ser agregados algunos filtros de calidad que comprueben que los mRNAs pertenezcan a cítricos y, del mismo modo, que las enzimas pertenezcan a cítricos (algunas entradas con elementos pertenecientes al ser humano y por tanto erróneos). Puede verse en el Fragmento de código A.28 . A continuación, en la Figura 3.23, puede verse un ejemplo de los datos escritos en el fichero `ortholog_relations.py`. Se obtienen 6.645 relaciones (0.250 MB de datos).

(1) `ec:1.1.1.284;K00121;enz_orth`
(2) `Ciclev10001399m.g.v1.0;K00121;gene_orth`

Figura 3.23: Ejemplo ficheros generados con ortholog_relations.py

El quinto script se encarga de generar las relaciones entre los pathways y las enzimas en los cuales estos intervienen (Tabla 3.39).

pathway_enzyme.py

Entrada	**Salida**
Cclementina_v1.0_KEGG-pathways.txt.gz kegg_txt_20110915_1636.txt.gz	pathway_enzyme.csv
Cabeceras de los ficheros	
(1):`END_ID(PATHWAY-ID);:START_ID(ENCIMA-ID);:TYPE`	

Tabla 3.39: Datos del script pathway_enzyme.py

Como hemos visto anteriormente, el caso de los pathways es un tanto especial ya que al provenir de dos archivos distintos pueden tener identificadores distintos. Por ello, se ha utilizado la descripción del pathway, que sí coincide, para unificar los identificadores y utilizar como identificador único y final la concatenación de ambos. A continuación, en la Figura 3.24, puede verse un ejemplo de los datos escritos en el fichero `pathway_enzyme.py`. Se obtienen 3.038 relaciones (0.1 MB de datos).

```
map00312;ec:3.5.2.6;intervenes
```

Figura 3.24: Ejemplo fichero generado con pathway_enzyme.py

Por último, el sexto script genera las relaciones entre los elementos cromosómicos y las variaciones. Es, junto con `variations.py` el script que más recursos consume y que más tiempo tarda en ejecutarse ya que debe iterar sobre todos los ficheros VCF (Tabla 3.40).

variation_elements.py

Entrada	**Salida**
Cclementina_182_v1.0.gene_exons.gff3.gz prueba_\w_(indel|snp)\.vcf\.gz[18]	variation_elements.csv
Cabeceras de los ficheros	
(1):START_ID(VARIATION-ID);:END_ID(ELEMENT-ID);:TYPE	

Tabla 3.40: Datos del script variation_elements.py

Comparte aspectos estructurales del script `variations.py` puesto que utiliza procesos para reducir el tiempo de ejecución y contiene algunos métodos como `io_loop` ya explicados anteriormente. No obstante tiene sus particularidades en cuanto a cómo procesa los datos:

En primer lugar, ejecuta el método `optimize`. Este método organiza los elementos cromosómicos en tres diccionarios:

- *scaffold_gene*: tiene como clave un scaffold y como valor la lista de genes que se encuentran en dicho scaffold.
- *gene_mrna*: tiene como clave un gen y como valor la lista de mRNAs que se encuentran en dicho gen.
- *mrna_element*: tiene como clave un mRNA y como valor la lista de exones, 5' cap, CDS y UTR 3' que se encuentran en dicho mRNA.

Una vez se han obtenido los diccionarios de datos, se inicia el método `io_loop` cuya funcionalidad no ha cambiado: si hay registros para escribir en ficheros los escribe.

[18] expresión regular debido al largo gran número de ficheros. Los valores del campo \w pueden encontrarse en la figura 3.5

A continuación, itera sobre los ficheros VCF ejecutando el método `load_n_write` que crea un *reader* del módulo `pyVCF` para los ficheros de modo que facilite la extracción de datos de las variaciones, las ordena por su cromosoma y, genera nuevos procesos (hasta un máximo de treinta y dos) que reciben bloques de 65.536 variaciones para procesarlas (método `fork_process`). Este método crea y añade al *pool* de procesos nuevos procesos que ejecutan el método `load_n_write_process`. Este método es el encargado de crear las relaciones entre las variaciones y sus elementos cromosómicos y los agrega a la cola compartida entre procesos para que puedan ser escritos.

En la Figura 3.25, puede verse un ejemplo de los datos escritos en el fichero `variation_elements.py`. Se obtienen 67.369.055 relaciones (4.1 GB de datos).

```
inde-1-21640504-ATATATAT-[A];Ciclev10010152m.g.v1.0;variation_in_gene
```

Figura 3.25: Ejemplo fichero generado con variation_elements.py

Durante el desarrollo de estos scripts se han realizado diversas iteraciones con el objetivo de reducir el tiempo de ejecución al mínimo. La primera iteración en realizar la transformación de datos de manera completa tuvo una duración de 68 horas. De esta primera versión se han realizado todo un conjunto de mejoras que han permitido ir reduciendo el tiempo total de ejecución de los scripts. Por orden cronológico son:

- Inicialmente, en el script `variation_elements.py` existía un único diccionario de datos cuya clave era el scaffold y el valor el resto de elementos. Para no realizar más comprobaciones de las estrictamente necesarias, se sustituyó por tres diccionarios (tal y como se ha explicado anteriormente). Con esta nueva estructuración de los datos evitamos que si, por ejemplo, una variación no se encuentra en un gen, ya no compruebe si se encuentra en el resto de elementos que se encuentran dentro de dicho gen; con la versión anterior, tan sólo filtraba por cromosoma.

- Se sustituye la forma en que se comprueban los valores duplicados en el script `variations.py`: se pasa de tener tres *sets* (uno para la variación y el gen, uno para la variación y el mRNA y otro para la variación y el resto de elementos) de datos a uno con el identificador de la variación. El resultado de esta modificación es que si una variación está duplicada evitamos procesarla como se hacía inicialmente donde una vez se había procesado la variación se comprobaba si las relaciones ya existían.

- Se comprimen los ficheros para reducir el tamaño de datos a leer. Una vez comprimido el fichero, tan solo hay que cambiar `with open(self.file, 'rt') as file` por `with gzip.open(self.file, 'rt') as file`.

- Indexado de los ficheros VCF mediante la herramienta tabix. En el Fragmento de código A.29 se muestran las líneas añadidas para trabajar con trabix dentro de los scripts.

- Se modifica el script para que utilice multiprocesamiento. Se permite que los ficheros sean analizados de manera paralela, de modo que las variaciones se reparten entre un conjunto de procesos los cuales extraen los datos y los

agregan a una cola. El proceso principal se mantiene a la espera de que en la cola haya un número determinado de datos a escribir y en ese momento los escribe y elimina de la cola.

– La comprobación de elementos repetidos se mueve fuera del script. Antes de realizar este cambio, el script mantenía un objeto donde almacenaba las variaciones procesadas, de modo que en cada nueva variación a procesar se comprobaba si ya se había procesado anteriormente. En caso afirmativo se ignoraba y se pasaba a la siguiente, en caso negativo se procesaba y se añadía al conjunto. Esta comprobación se ha eliminado y ahora es, tras la ejecución del script, cuando se eliminan los valores repetidos de los ficheros (utilizando la herramienta awk): `time awk '!x[$0]++'` ... Por ejemplo, en catorce minutos elimina los elementos repetidos del fichero con el mayor número de elementos mientras que en la reducción de tiempos hablamos de horas.

En la Tabla 3.41 puede observarse la evolución de la duración de ejecución del script `create_model.sh` con forme se han ido aplicando las mejoras

Mejora	**Tiempo de ejecución**
Original	68h.
Aumento del número de diccionarios en `variation_elements.py`	66h.
Reducción del número de diccionarios en `variations.py`	64h.
Compresión de ficheros	61h.
Indexado con tabix	59h.
Multiproceso	20h.
Uso de awk para eliminar duplicados	15h.

Tabla 3.41: Repercusión de la aplicación de mejoras en el tiempo de ejecución del script `create_model.sh`

En total se ha conseguido reducir el tiempo de ejecución hasta llegar a un 22 % del tiempo original. La mayor reducción de tiempo se ha obtenido con el multiprocesamiento, reduciendo en un tercio el tiempo de ejecución (de 59 horas a 20) seguido del uso de la herramienta awk para el borrado de elementos duplicados.

Finalmente, una vez se han realizado todos los scripts necesarios y han sido optimizados en múltiples iteraciones de refactorización, solo resta ejecutarlos para obtener los ficheros CSV necesarios para crear la base de datos. Con este objetivo, se ha creado un script en `bash` que ejecute todos los scripts así como los comandos necesarios para eliminar datos duplicados en los ficheros resultantes. En el Fragmento de código A.6 se muestran las llamadas a los scripts así como a la herramienta `awk` para eliminar los valores duplicados. El tiempo total de ejecu-

ción de este script es de dieciséis horas tras las cuales el proceso de carga de los datos en Neo4J es directo.

3.3.5. Proceso de Carga

Una vez se disponen de todos los ficheros CSV y sus correspondientes cabeceras se genera la base de datos mediante la herramienta `neo4j-admin`[19]. Es la principal herramienta para la gestión de Neo4J y mediante el parámetro `import` se puede generar una base de datos partiendo de ficheros CSV. Existen dos formas de importar datos en Neo4J, la segunda es mediante la cláusula `IMPORT CSV` del `cypher` aunque está pensada para conjuntos de dato de tamaño pequeño o mediano.

La herramienta obtiene los ficheros de un directorio definido en el fichero de configuración que, en nuestro caso, se encuentra en la ruta `/etc/neo4j/neo4j.conf`. Existen dos posibilidades: la primera es colocar los ficheros en la ruta especificada, la segunda modificar dicha ruta en la configuración. Utiliza el campo ID para generar correctamente las relaciones entre nodos, que se etiquetan con los valores proporcionados en el último campo de los nodos. A continuación, se explican los parámetros utilizados en la ejecución de la herramienta:

- **database**: Nombre de la base de datos creada
- **delimiter**: Carácter que delimita los valores dentro de los ficheros
- **array-delimiter**: Carácter que delimita elementos dentro de un campo de tipo *array*
- **ignore-duplicate-nodes**: indica si los nodos duplicados deberían ser ignorados durante la generación de la base de datos
- **id-type**: Tipo de los identificadores de los nodos
- **nodes**: fichero de cabecera y de datos de cada tipo de nodo existente
- **relationships**: fichero de cabecera y de datos de cada tipo de relación existente

A continuación, en el Fragmento de código A.5 se puede ver el script de creación de la base de datos (algunos parámetros --nodes y --relationships han sido eliminados ya que tan sólo cambia el nombre del fichero). Como se puede observar en el Fragmento de código A.7 , la generación de la base de datos se realizó en veintitrés minutos y ha generado 278.082.816 nodos (diecisiete tipos de nodos diferentes), 465.146.788 relaciones (treinta y una tipos de relaciones distintas) y 1.743.968.582 propiedades (en nodos y relaciones). En total la base de datos ocupa 104.39 GB, de los cuales 4.87 son de nodos, 15.82 de relaciones, 71,5 de propiedades de tipo básico, 12.20 a propiedades de tipo array y el resto a ficheros de configuración y logs.

[19] https://neo4j.com/docs/operations-manual/current/tools/neo4j-admin/

3.3.6. Configuración

Una vez se ha creado la base de datos es indispensable modificar los parámetros de configuración de esta para su correcto funcionamiento. En la Tabla 3.42 pueden observarse los parámetros modificados y sus valores finales.

Opciones de Configuración

Parámetro	Descripción	Valor
`dbms.memory.heap.initial_size`	Tamaño inicial del *heap* de memoria utilizado por Neo4J para manejar las operaciones (ejecución de consultas, estados de las transacciones, gestión del grafo, etc.)	40g
`dbms.memory.heap.max_size`	Tamaño máximo del *heap* de memoria utilizado por Neo4J.	40g
`dbms.memory.pagechache.size`	Especifica cuanta memoria se le permite utilizar a Neo4J para *caching*. Se utiliza para cachear tanto los datos como los índices en memoria con el objetivo de mejorar el rendimiento. Debería estar, si no toda, casi toda la información cacheada en memoria para optimizar el rendimiento	300g
`dbms.connectors.default_listen_address`	Permite que se reciban conexiones remotas a la base de datos	0.0.0.0

Tabla 3.42: Opciones de configuración de la base de datos modificadas

Con el objetivo de mejorar el rendimiento de las consultas, se han creado índices sobre las siguientes propiedades:

- Atributo `annotation_impact` en los nodos de tipo `annotation`.
- Atributo `Variety` en los nodos de tipo `lecture`.
- Atributo `pos_abs` en los nodos de tipo `Variation`.

Finalmente, se ha modificado el planificador del sistema operativo, debido a la forma de trabajar de la base de datos y al uso que realizan sus usuarios que requiere de muchas más lecturas que escrituras. El algoritmo típicamente utilizado, *Completely Fair Queuing* (CFQ)[20] no es el adecuado puesto que, al ser su prioridad maximizar el uso de CPU, las lecturas no tendrán prioridad por encima de las escrituras. Con esto en mente, es una buena opción cambiar al planificador *deadline*, que da prioridad a las lecturas por encima de las escrituras, procesándolas lo antes posible.

[20] proporciona un equilibrio entre rendimiento y latencia.

Como resultado, aumenta la latencia de las escrituras, pero disminuye la de las lecturas. En el Fragmento de código A.8 se observa cómo se ha cambiado el planificador del sistema operativo. Además, se ha deshabilitado (mediante la modificación del fichero de la tabla del sistema operativo (*fstab*)) la actualización de la fecha de último acceso de los ficheros, de modo que no se reescribirán los metadatos de los ficheros en las escrituras mejorando la eficiencia.

3.4 Backend

Esta sección trata sobre el *backend* de la aplicación y consta de dos subsecciones. En la primera (Sección 3.4.1), contiene la especificación de la API del servidor. La segunda (Sección 3.4.2), detalla cómo ha sido implementado el *backend*.

3.4.1. API

En esta subsección se detalla la API diseñada. Por cada punto de entrada se definen una serie de campos con el objetivo de eliminar cualquier tipo de ambigüedad. Los campos definidos son los siguientes:

- Nombre: nombre de la ruta.
- Descripción: descripción de la funcionalidad de la ruta y requisitos que cumple al ser implementada.
- Cabeceras: cabeceras necesarias para la petición.
- Parámetros: parámetros necesarios para poder realizar la consulta.
- Respuesta: respuesta que proporciona ante una petición exitosa.
- Errores: errores definidos para la ruta.

Todas las rutas comparten las mismas cabeceras de las peticiones que pasa a ser descrito a continuación. Únicamente se requiere identificar el tipo del recurso tal y como se observa en la Tabla 3.43.

Cabeceras de la petición		
`Content-Type`	String	Indica el tipo del recurso
Ejemplo		
{ “Content-Type”: “application/json” }		

Tabla 3.43: Información común de las consultas

Basándonos en los requisitos detallados en la Sección 3.2 se han definido los siguiente puntos de entrada:

Consulta 1 — /q1

Implementa el requisito R0 y R1.0. Recibe una serie de parámetros y obtiene las variaciones presentes en A y no en B que cumplen los criterios establecidos. La petición consta de siete parámetros descritos en la Tabla 3.44.

Parámetros

Campo	Tipo	Descripción
`A`	String[]	Primer grupo de variedades. Se utiliza como conjunto de variedades que deben presentar las variaciones.
`B`	String[]	Segundo grupo de variedades. Se utiliza como conjunto de variedades que no deben presentar las variaciones.
`filter` *opcional*	String	Identifica el filtro de calidad que se desea aplicar para filtrar las variaciones. Valor por defecto: lecture
`annotation_impact` *opcional*	String[]	Lista de impactos sobre los genes que las variaciones deben poseer. Valor por defecto: [“HIGH”]
`min` *opcional*	Number	Indica en cuantas variedades de A debe aparecer como mínimo una variación para no ser descartada. Valor por defecto: Longitud de A Tamaño de rango: 1 − Longitud de A
`max` *opcional*	Number	Indica en cuantas variedades de B puede aparecer como máximo una variación para no ser descartada Valor por defecto: 0 Tamaño de rango: 0 − Longitud de B
`depth` *opcional*	Number	Indica con qué nivel de profundidad (desde un punto de vista genómico) se ejecutará la consulta. 0 para variaciones en general, 1 para variaciones que se encuentran posicionalmente un gen, 2 para variaciones que se encuentran posicionalmente en un exón, 3 para aquellas que estén dentro de un intrón, 4 para UTR y 5 para CDS. Valor por defecto: 5 Tamaño de rango: 0 − 5

Continúa en la página posterior

Continúa de la página anterior

Campo	Tipo	Descripción
`AB` *opcional*	Object[]	Lista de uno o más objetos cada uno de los cuales contiene dos atributos; op y *value*. El campo op puede recibir los valores g (mayor), ge (mayor o igual), l (menor), le (menor o igual), b (entre dos valores excluidos) y be (entre dos valores incluidos). El campo *type* contiene un valor entero si el campo op es g, ge, l o le y un array con dos enteros si el campo op es b o be.

Ejemplo

```
{
  ‘‘A’’: ["000","003", "004", "005", "006", "007", "008", "009", "010"],
  ‘‘B’’: ["900","901","902","903","904","905"],
  ‘‘filter’’: "10-70",
  ‘‘annotation_impact’’: ["HIGH"],
  ‘‘min’’: 9,
  ‘‘max’’: 0,
  ‘‘depth’’: 4,
  ‘‘AB’’: [{op:"ge", "value": 0.8}]
}
```

Tabla 3.44: Parámetros de la consulta q1

En el ejemplo se han pedido aquellas variaciones presentes las nueve variedades presentes en el grupo “A” y que no estén presentes en ninguna de las seis variedades del grupo “B”. Las variaciones deben tener un valor DP mayor que 10, un valor GQ mayor que 70 y un valor AB mayor o igual a 0.8. Además, debe tener una anotación con impacto alto (*HIGH*) y debe estar posicionalmente dentro de un intrón.

Si la consulta se procesa correctamente la respuesta consta de dos campos, descritos en la Tabla 3.45.

Parámetros

Campo	Tipo	Descripción
`count`	Number	Número total de variaciones que cumplen los criterios especificados en la consulta. Este número puede diferir de la longitud del array variations porque las variaciones no están agrupadas por gen, por lo que si una variación afecta a más de un gen aparecerá dos veces.
`variations`	String[]	Array con la lista de variaciones encontradas y su información.

Continúa en la página posterior

Continúa de la página anterior

Ejemplo


```
{
  ‘‘count’’: 74,
  ‘‘variations’’ {
    ‘‘v’’: ‘‘snp-6-25583447-A-[G]’’,
    ‘‘annotation’’: [ ‘‘splice_donor_variant’’ ],
    ‘‘gene’’: ‘‘Ciclev10012200m.g.v1.0’’,
    ‘‘go’’: [{
      ‘‘type’’: ‘‘Molecular function‘‘, ‘‘description’’: ‘‘binding’’,
      ‘‘id’’: ‘‘0005488’’
    }],
    ‘‘domain’’: [{
      ‘‘description’’: ‘‘Mitochondrial carrier protein’’, ‘‘id’’: ‘‘IPR002067’’
    }],
    ‘‘enzyme’’: [{
      ‘‘n_seq’’: { ‘‘low’’: 5, ‘‘high’’: 0 }, ‘‘id’’: ‘‘ec:1.1.1.34’’,
      ‘‘description’’: ‘‘hydroxymethylglutaryl-CoA reductase (NADPH)’’}
    }],
    ‘‘pathway’’: [{
      ‘‘description’’: ‘‘terpenoid backbone biosynthesis’’,
      ‘‘id_map’’: ‘‘map00900’’, ‘‘id_ko’’: ‘‘ko00900’’
    }],
    ‘‘varieties’’: [ ‘‘003’’ ]
  }
}
```


Tabla 3.45: Parámetros de la respuesta de la consulta q0

El array *variations* de la respuesta contiene la lista de variaciones encontradas. Cada variación consta de ocho atributos. El primer atributo, llamado ”v“ es el identificador de la variación, formado por el tipo de variación, el *scaffold* y posición donde se encuentra y los alelos de referencia y alternativo. El segundo atributo, llamado ”annotation“ es un array que contiene una lista con los efectos que provoca la variación en el gen al que afecta. El tercer atributo, llamado ”gen“, contiene el identificador del gen al cual afecta la variación. Si una variación afecta a más de un gen aparecerá tantas veces en el array como genes afecte. El cuarto atributo, llamado ”go“ es un array con una lista nodos de tipo GO que se relacionan con las proteínas codificadas por el gen al que la variación afecta donde cada elemento del array contiene los atributos *type*, *description* e *id*. El quinto atributo, llamado ”domain“ contiene un array con los dominios asociados a las proteínas codificadas por el gen afectado por la variación. Cada dominio se compone de los campos *description* e *id*. El sexto atributo, llamado ”enzime“ contiene una lista con las enzimas codificadas por el gen afectado por la variación. De cada enzima se dispone de los atributos n_seq, *description* e *id*. El séptimo atributo, llamado ”pathway“, contiene una lista con los *pathways* afectados por las enzimas codificadas por el gen afectado por la variación. De cada *pathway* se dispone de los

atributos *id_map*, *id_ko* y *description*. Por último, el octavo atributo, llamado "varieties"contiene una lista con los códigos de las variedades donde se encuentra presente la variación. Es especialmente útil si el parámetro min de la petición es menor que el número de variedades del conjunto A.

Si la consulta ha producido algún tipo de error, este será enviado de vuelta al usuario. Dependiendo del fallo, el código y mensaje del error cambiará. En la Tabla 3.46 puede verse una lista completa con los posibles errores. Se han definido ocho errores relacionados con errores en el formato del mensaje enviado, un error específico relativo a la imposibilidad de que el servidor se conecte a la base de datos y un error genérico para todas las demás posibilidades.

Errores

Código	Nombre	Descripción
400	variedad inválida en A	Se ha utilizado un código de variedad inválido en el conjunto A
400	variedad inválida en B	Se ha utilizado un código de variedad inválido en el conjunto B
400	Conjunto A vacío	El conjunto de variedades A debe tener al menos una variedad
400	impacto de la anotación inválido	Se ha utilizado un impacto de anotación inválido
400	filtro de calidad inválido	Se ha utilizado un filtro de calidad inválido
400	campo min inválido	Se ha utilizado un valor inválido para el campo min
400	campo max inválido	Se ha utilizado un valor inválido para el campo max
400	campo depth inválido	Se ha utilizado un valor inválido para el campo depth
400	error	Base de datos no disponible
500	error	Algo falló

Tabla 3.46: Respuestas con errores de la consulta q0

Consulta 2 – /by_pos

Similar a la consulta 1, recibe una serie de parámetros y obtiene las variaciones presentes en A y no en B que cumplen los criterios establecidos. La diferencia radica en que esta consulta no utiliza la profundidad posicional como un filtro. En su lugar, se vale de dos atributos nuevos: genes y bases. Filtra las variaciones que se encuentren posicionalmente dentro de los genes indicados o a un número de bases menor al definido en el parámetro bases. La petición consta de ocho parámetros descritos en la Tabla 3.47.

Parámetros

Campo	Tipo	Descripción
`A`	String[]	Primer grupo de variedades. Se utiliza como conjunto de variedades que deben presentar las variaciones.
`B`	String[]	Segundo grupo de variedades. Se utiliza como conjunto de variedades que no deben presentar las variaciones.
`filter` *opcional*	String	Identifica el filtro de calidad que se desea aplicar para filtrar las variaciones. Valor por defecto: lecture
`annotation_impact` *opcional*	String[]	Lista de impactos sobre los genes que las variaciones deben poseer. Valor por defecto: [“HIGH”]
`min` *opcional*	Number	Indica en cuantas variedades de A debe aparecer como mínimo una variación para no ser descartada. Valor por defecto: Longitud de A Tamaño de rango: $1 -$ Longitud de A
`max` *opcional*	Number	Indica en cuantas variedades de B puede aparecer como máximo una variación para no ser descartada Valor por defecto: 0 Tamaño de rango: $0 -$ Longitud de B
`genes`	String[]	Indica los genes de los cuales se van a obtener las posiciones de inicio y fin para filtrar las variaciones
`bases`	Number	Indica el número de bases que una variación puede estar alejada de las posiciones establecidas por los genes sin que la variación sea desechada
`AB` *opcional*	Object[]	Lista de uno o más objetos cada uno de los cuales contiene dos atributos; op y *value*. El campo op puede recibir los valores g (mayor), ge (mayor o igual), l (menor), le (menor o igual), b (entre dos valores excluidos) y be (entre dos valores incluidos). El campo *type* contiene un valor entero si el campo op es g, ge, l o le y un array con dos enteros si el campo op es b o be.

Ejemplo

```
{
  ‘‘A’’: ["000","003", "004", "005", "006", "007", "008", "009", "010"],
  ‘‘B’’: ["900","901","902","903","904","905"],
  ‘‘filter’’: "10-70",
```

Continúa en la página posterior

Continúa de la página anterior

Campo	Tipo	Descripción

```
  ‘‘annotation_impact’’: ["HIGH"],
  ‘‘min’’: 9,
  ‘‘max’’: 0,
  ‘‘genes’’: [’Ciclev1000236m.g.v1.0’],
  ‘‘bases’’: 5000,
  ‘‘AB’’: [{op:"ge", "value": 0.8}]
}
```

Tabla 3.47: Parámetros de la consulta by_pos

En el ejemplo se han pedido aquellas variaciones presentes las nueve variedades presentes en el grupo “A” y que no estén presentes en ninguna de las seis variedades del grupo “B”. Las variaciones deben tener un valor DP mayor que 10, un valor GQ mayor que 70 y un valor AB mayor o igual a 0.8. Además, debe tener una anotación con impacto alto (*HIGH*) y debe estar posicionalmente dentro del gen “Ciclev1000236m.g.v1.0” o como mucho a quinientas pares de bases de distancia de dicho gen.

El formato de respuesta es el mismo que con la Consulta 1 y puede ser consultado en la Tabla 3.45. Respecto a los errores, son los mismos que en la primera consulta (Tabla 3.46) salvo con la diferencia de que se sustituyó el error del campo depth por dos nuevos mensajes de error, uno por cada nuevo atributo incluido en la consulta: genes y bases.

Consulta 3 — /ge

Esta consulta obtiene los genes relacionados dada una anotación funcional. Se pueden elegir tres tipos de anotaciones: enzima, pathway y dominio. La petición consta de dos parámetros, siendo el segundo de ellos opcional (Tabla 3.48).

Parámetros

Campo	Tipo	Descripción
`type`	String	Tipo de anotación funcional.
`id` *opcional*	String	Identificador de la anotación. Opciones: enzima, pathway y dominio

Ejemplo

```
{
  ‘‘type’’: pathway,
  ‘‘id’’: steroid biosynthesis,
}
```

Tabla 3.48: Parámetros de la consulta ge

En caso de incluirse el campo id en la consulta, la respuesta presenta la estructura de la Tabla 3.49.

Parámetros		
Campo	**Tipo**	**Descripción**
`result`	Object[]	Anotación funcional seleccionada y los genes con los cuales se relaciona

Ejemplo

```
{
  ‘‘result’’:
    [
      "steroid biosynthesis":
        [
          ‘‘Ciclev1000234.g.v1.0’’,‘‘Ciclev100986.g.v1.0’’
        ]
    ]
}
```

Tabla 3.49: Parámetros de la respuesta de la consulta q0

En el ejemplo se han pedido aquellos genes que estén relacionados con una anotación funcional de tipo *pathway* e identificador “steroid biosynthesis”.

En caso contrario, el formato es el mismo pero el array en vez de constar de un único elemento incluye todas las anotaciones funcionales con los genes relacionados.

Consulta 4 — /reg_chr

Esta consulta recibe un rango posicional, así como un conjunto de filtros de calidad y devuelve aquellas variaciones que se encuentren dentro del rango definido y que cumplan los filtros establecidos (Tabla 3.50).

Parámetros		
Campo	**Tipo**	**Descripción**
`scaffold`	Number	Identifica el scaffold donde se establecerá el rango de posiciones para realizar la búsqueda de variaciones.
`start_pos`	Number	Posición de inicio del rango de búsqueda.
`end_pos`	Number	Posición de fin del rango de búsqueda.
`filter` *opcional*	String	Identifica el filtro de calidad que se desea aplicar para filtrar las variaciones. Valor por defecto: lecture
`annotation_impact` *opcional*	String[]	Lista de impactos sobre los genes que las variaciones deben poseer. Valor por defecto: [“HIGH”]

Continúa en la página posterior

Continúa de la página anterior

Campo	Tipo	Descripción
AB *opcional*	Object[]	Lista de uno o más objetos cada uno de los cuales contiene dos atributos; op y *value*. El campo op puede recibir los valores g (mayor), ge (mayor o igual), l (menor), le (menor o igual), b (entre dos valores excluidos) y be (entre dos valores incluidos). El campo *type* contiene un valor entero si el campo op es g, ge, l o le y un array con dos enteros si el campo op es b o be.

Ejemplo

```
{
  ‘‘scaffold’’: 1,
  ‘‘start_pos’’: 25000000,
  ‘‘end_pos’’: 26000000,
  ‘‘filter’’: "10-70",
  ‘‘annotation_impact’’: ["HIGH"],
  ‘‘AB’’: [{op:"ge", "value": 0.8}]
}
```

Tabla 3.50: Parámetros de la consulta reg_chr

En el ejemplo se han pedido aquellas variaciones presentes posicionales en el *scaffold* 1, entre las posiciones 25.000.000 y 26.000.000. Deben tener un valor DP mayor que 10, un valor GQ mayor que 70 y un valor AB mayor o igual a 0.8. Además, debe tener una anotación con impacto alto (*HIGH*) y debe estar posicionalmente dentro del gen “Ciclev1000236m.g.v1.0” o como mucho a quinientas pares de bases de distancia de dicho gen.

El formato de respuesta es el mismo que con la Consulta 1 y puede ser consultado en la Tabla 3.45. Respecto a los errores, son los mismos que en la primera consulta (Tabla 3.46) sustituyendo los errores de los campos eliminados por los nuevos: scaffold, start_pos y end_pos.

Consulta 5 — /fc

Esta consulta proporciona el conjunto de filtros de calidad definidos en la base de datos. Es la única petición de tipo `GET`. La respuesta consta de una lista con los filtros de calidad definidos como puede verse en la Tabla 3.51.

Parámetros		
Campo	**Tipo**	**Descripción**
Ejemplo		

```
[
  '10-60','10-70','10-80','10-90'
]
```

Tabla 3.51: Respuesta de la consulta fc

Filtros de calidad — /updating

Implementa nuevos filtros de calidad añadiendo la etiqueta correspondiente a los nodos de tipo lectura (Tabla 3.52).

Parámetros		
Campo	**Tipo**	**Descripción**
DP	Number	Valor DP utilizado para el filtro
GQ	Number	Valor GQ utilizado para el filtro
Ejemplo		

```
{
  ‘‘DP’’:100
  ‘‘GQ’’:100
}
```

Tabla 3.52: Parámetros de la respuesta de la respuesta updating

3.4.2. Implementación

En esta subsección se detalla el *backend* implementado para la aplicación web. Se ha utilizado el entorno de ejecución de JavaSript Node.js[21], el módulo *express*[22] como *framework* de aplicación web y *Auth0*[23] para la gestión de usuarios y autenticación. A continuación, se identifican cada uno de los directorios que contiene el proyecto como una primera aproximación general para, a partir de esta, entrar en mayor nivel de detalle.

- certs: directorio donde se alojan los certificados utilizados para realizar una navegación segura.

[21] https://nodejs.org/es/
[22] http://expressjs.com/es/
[23] https://auth0.com/

- docs: en este directorio se almacena la documentación generada automáticamente con el uso de los módulos esdoc[24] (para documentar el código y generar el manual) y apidoc[25] (para documentar el API)

- logs: almacena los registros de la aplicación. Múltiples eventos son almacenados en los registros, que se organizan según el script del que provienen. Por un lado, se almacenan todos los accesos a la aplicación. Por otro, las distintas llamadas al API almacenando los parámetros de consulta, la respuesta, el tiempo de respuesta y los posibles errores que puedan ocurrir. De manera automática los registros son comprimidos cada 24 horas añadiendo la fecha del registro al nombre del fichero comprimido.

- manual: documentación en formato markdown[26] que es incluido en la documentación cuando esta es generada. Consta de tres partes: la primera contiene información sobre cómo instalar las dependencias tecnológicas del proyecto y cómo descargar y configurar el repositorio. La segunda parte contiene información sobre cómo utilizar el servidor, es decir, cómo iniciarlo en modo desarrollo o en modo producción, cómo generar la documentación y cómo ejecutar los tests. Por último, la tercera parte contiene información acerca de cómo modificar la configurar tanto del servidor como de acceso a la base de datos.

- node_modules: almacena todos los módulos incluidos en el fichero `package.json`.

- src: contiene todo el código fuente de la aplicación. A continuación se explicará en mayor detalle el contenido de esta carpeta.

- tests: incluye los tests generados para validar la funcionalidad de la aplicación. Hace uso del módulo jest[27]

En este fichero se almacena toda la información relativa a la configuración del servidor. Esta configuración varia en función del entorno en que se ejecuta el servidor: desarrollo o producción. La información almacenada es la siguiente:

- Dirección y credenciales de acceso a la base de datos.

- Ruta en la cual se desplegará el servidor y de almacenamiento de los *logs*.

- Información relativa a la configuración de auth0.

- Direcciones de red de origen aceptadas.

La creación de la aplicación web se realiza en el fichero `app.js`. En él se instancia la aplicación, se configuran las rutas de acceso a los servicios de la API, y se configuran los *middlewares* que utilizará. En la siguiente lista se muestran todos los *middlewares* utilizados:

[24] https://esdoc.org/
[25] http://apidocjs.com/
[26] https://daringfireball.net/projects/markdown/
[27] https://facebook.github.io/jest/

- Morgan: monitoriza y registra todos los accesos al servidor.
- Helmet: aumenta la seguridad de la aplicación al incluir cabeceras a las respuestas que responde el servidor.
- Ip_filter: comprueba que la IP desde la cual se realiza una petición esté incluida dentro de la lista de IPs válidas.
- BodyParser: modifica el formato del cuerpo de la petición para que sea JSON.
- CookieParser: modifica el formato de las *cookies* de la petición para que sea JSON.
- Compression: comprime la respuesta para minimizar el uso de red.
- Jwt: comprueba que las peticiones se realizan con un *token* y que éste es válido.

A continuación, analizamos el contenido de los paquetes presentes en el directorio src. Se encuentran tres paquetes: bin, lib, public y routes: El fichero `www.js` lanza el servidor y obtiene el diccionario de datos (fichero `dataDictionary.js`). Actúa como punto de entrada a la aplicación.

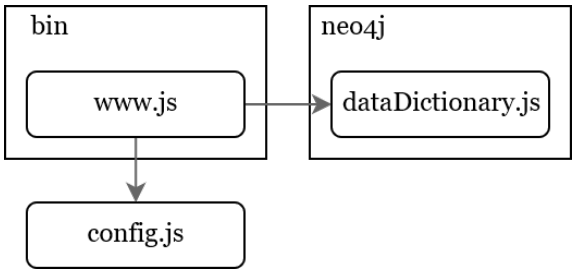

Figura 3.26: Diagrama de componentes del paquete src con sus dependencias

El directorio bin (Figura 3.26) consta de un único fichero llamado `www.js`. Este script se encarga de lanzar el servidor. Dependiendo de si se lanza en producción o desarrollo utiliza unas rutas y puerto diferente. En el proceso de configuración ejecuta la función `dataTable` del script `dataDictionary.js` (Fragmento de código A.14 , para más detalles véase página 91).

El directorio logging (Figura 3.27) contiene el fichero `loggerFunctions.js`. Este fichero se encarga de proveer una *logger* parametrizado a cada uno de los componentes. El *logger* almacenará los eventos de cada uno de los componentes de manera independiente.

El directorio neo4j (Figura 3.28) consta de diez ficheros: `configureDriver.js`, `dataDictionary.js`, `domain.js`, `enzymes.js`, `generators.js`, `genes.js`, `pathway.js`, `queryRunner.js`, `recordFunctions.js` y `variations.js`. Este paquete contiene la funcionalidad necesaria para generar consultas a la base de datos, realizar estas consultas y modificar el formato del resultado obtenido.

El fichero `configureDriver.js` se encarga de gestionar la creación de un `driver` utilizando las credenciales del fichero `config.js` y lo exporta para que pueda ser utilizado con el resto de funciones.

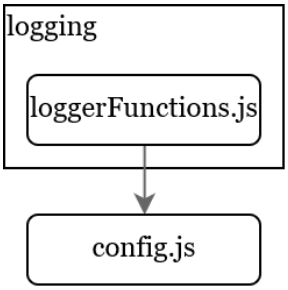

Figura 3.27: Diagrama de componentes del paquete logging con sus dependencias

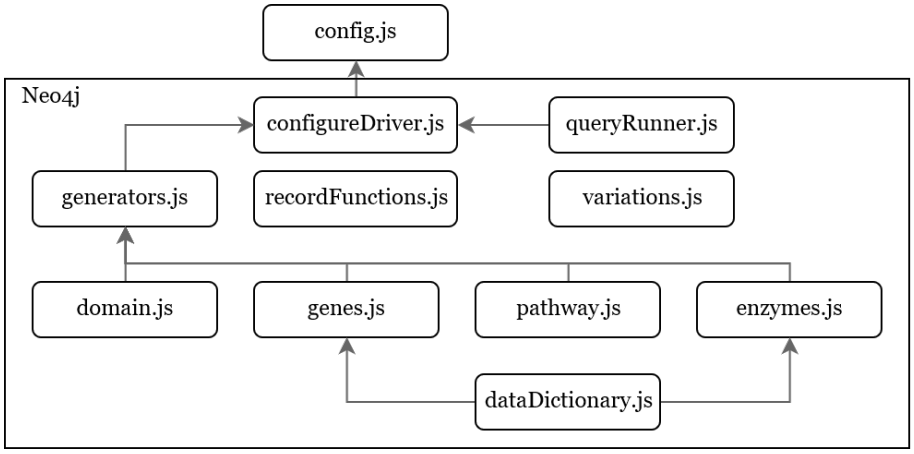

Figura 3.28: Diagrama de componentes del paquete neo4j con sus dependencias

El fichero `dataDictionary.js` genera un diccionario de relaciones entre el elemento gen y el resto de elementos con los que se relaciona. Este diccionario de datos se obtiene al instanciar el servidor y se utiliza para complementar la información obtenida por las consultas realizadas a la base de datos. Como se observa en el Fragmento de código A.15 , el fichero ejecuta seis funciones para obtener las relaciones entre los genes y los elementos GO, dominio, proteína, grupo ortólogo, enzima y pathway.

El fichero `generators.js` exporta una función parametrizable que genera diccionarios de datos para nodos directamente conectados. Como se puede observar en el Fragmento de código A.16 , la función toma tres parámetros: "x", que actuará como elemento usado para la clave del diccionario, "y", que identifica la relación de unión entre los dos nodos, "z" que actuará como conjunto de valores del diccionario e "id", que identifica el atributo del elemento "x" utilizado como clave. Tomando estos cuatro parámetros se ejecuta la consulta en la base de datos. Mediante en uso de *streams*, por cada uno de los elementos obtenidos obtiene los atributos de interés.

El fichero domain.js implementa la función paramétrica `XCollectY` del fichero `generators.js`. Los parámetros con los que se instancia la función son: para el parámetro x: *'domain'*, para el parámetro y: *'associated'*, para el parámetro z: *'gene'* y para el parámetro id: *'id'*. De este modo, se obtiene un diccionario cuya clave es el identificador de un dominio y valor una lista con los genes asociados a dicho dominio.

El fichero enzymes.js implementa la función paramétrica `XCollectY` del fichero `generators.js` dos veces. Para la primera función, los parámetros con los que se instancia la función son: para el parámetro x: *'enzyme'*, para el parámetro y: *'gene_encodes_enzyme'*, para el parámetro z: *'gene'* y para el parámetro id: *'id'*. Para la segunda función, los parámetros con los que se instancia la función son: para el parámetro x: *'enzyme'*, para el parámetro y: *'intervenes'*, para el parámetro z: *'pathway'* y para el parámetro id: *'id'*. De este modo, se obtienen dos diccionarios cuya clave es el identificador de una enzima y, para la primera función, el valor es una lista con los enzimas asociados a dicha gen y, para la segunda función el es valor una lista con los enzimas asociados a dicha pathway.

El fichero genes.js implementa la función paramétrica `XCollectY` del fichero `generators.js` cinco veces. Para la primera función, los parámetros con los que se instancia la función son: para el parámetro x: *'gen'*, para el parámetro y: *'associated'*, para el parámetro z: *'GO'* y para el parámetro id: *'id'*. Para la segunda función, los parámetros con los que se instancia la función son: para el parámetro x: *'gen'*, para el parámetro y: *'associated'*, para el parámetro z: *'domain'* y para el parámetro id: *'id'*. Para la tercera función, los parámetros con los que se instancia la función son: para el parámetro x: *'gen'*, para el parámetro y: *'gene_encodes_protein'*, para el parámetro z: *'protein'* y para el parámetro id: *'id'*. Para la cuarta función, los parámetros con los que se instancia la función son: para el parámetro x: *'gen'*, para el parámetro y: *'gene_orth'*, para el parámetro z: *'ortholog'* y para el parámetro id: *'id'*. Por último, para la quinnta función, los parámetros con los que se instancia la función son: para el parámetro x: *'gen'*, para el parámetro y: *'gene_encodes_enzyme'*, para el parámetro z: *'enzyme'* y para el parámetro id: *'id'*. De este modo, se obtienen cinco diccionarios cuya clave es el identificador de un gen y, para la primera función, el valor es una lista con los GO asociados a dicho gen, para la segunda función, el valor es una lista con los dominios asociados a dicho gen, para la tercera función, el valor es una lista con los proteínas asociados a dicho gen, para la cuarta función, el valor es una lista con los grupos ortólogos asociados a dicho gen y para la quinta función el valor es una lista con los enzimas asociados a dicho gen.

El fichero pathway.js implementa la función paramétrica `XCollectY` del fichero `generators.js`. Los parámetros con los que se instancia la función son: para el parámetro x: *'pathway'*, para el parámetro y: *'intervenes'*, para el parámetro z: *'enzyme'* y para el parámetro id: *'description'*. De este modo, se obtiene un diccionario cuya clave es el identificador de un pathway y valor es una lista con las enzimas asociadas a dicho pathway.

El fichero variations.js contiene las funciones encargadas de generar las consultas a la base de datos requeridas por la API. Define cuatro funciones:

QueryGenerator

Utilizado para las consultas por variedades (ruta `/q1` de la API). En base a los parámetros recibido genera automáticamente la consulta correspondiente. Los parámetros que recibe así como las subfunciones definidas para la generación de las consultas pueden consultarse en el Fragmento de código A.17 . En el Fragmen-

to de código A.10 puede observarse un ejemplo de consulta en lenguaje `cypher` generado por esta función.

get_gene_positions

Se trata de una consulta auxiliar para la función `get_variations_by_position`. Recibe dos parámetros: genes y bases, y lanza una consulta para obtener los valores que debe utilizar para filtrar las variaciones de manera que únicamente aparezcan aquellas presentes posicionalmente dentro del intervalo definido. EL resultado de dicha consulta es la posición de inicio del gen restándole el valor de las bases y la posición de fin del gen sumándole el valor de las bases. De este modo el rango de búsqueda va incluye las posiciones del gen y las posiciones cercanas deseadas. Estos valores se utilizan para filtrar por el índice posicional creado en las variaciones. En el Fragmento de código A.11 se observa un ejemplo real de consulta generada con esta función.

get_variations_by_position

Utilizado para las consultas por genes (ruta `/by_pos` de la API). Muy similar a la función `QueryGenerator`, cuenta con los mismos parámetros de entrada excepto al parámetro "depth", que es sustituido por el parámetro "positions". El parámetro "positions" contiene una lista de posiciones con las que filtrar las variaciones por su atributo "pos_abs". Los parámetros que recibe así como las subfunciones definidas para la generación de la consulta pueden consultarse en el Fragmento de código A.18 . En el Fragmento de código A.12 puede observarse un ejemplo de consulta en lenguaje `cypher` generado por esta función.

get_variations_by_chr_region

Utilizado para las consultas por región cromosómica (ruta `/reg_chr` de la API). Recibe seis parámetros: "scaffold", "start_pos", "end_pos", "filter", "annotation_impact" y "AB". Genera una consulta cuyo objetivo es obtener las variaciones que se encuentren dentro del rango posicional del *scaffold* indicado que cumplan con los filtros definidos. En el Fragmento de código A.19 pueden observarse los parámetros de la función así como las subfunciones definidas para la generación de la consulta. En el Fragmento de código A.13 puede observarse un ejemplo de consulta en lenguaje `cypher` generado por esta función.

El fichero `queryRunner.js` implementa dos funciones. La primera de ellas, "runOne" se encarga de ejecutar una consulta contra la base de datos. Mediante el uso de *streams*, va modificando cada uno de los registros que recibe y, al finalizar, incorpora una serie de información extra además del resultado: consulta ejecutada a la base de datos, número de elementos y tiempo de ejecución. El número de resultado es necesario pues identifica de manera unívoca el número de variaciones encontradas, que puede no coincidir con la longitud del *array* de resultado. Esto es debido a que si una variación afecta a más de un gen, aparecerá repetida variando el gen y las anotaciones funcionales correspondientes. En el Fragmento de código A.20 se muestran los fragmentos más relevantes de la

función. La segunda, "runMultiple" ejecuta las consultas que recibe llamando a la función `runOne` definida anteriormente.

Como acabamos de ver, las funciones definidas en el fichero `queryRunner.js` reciben como parámetro una función para ser ejecutada sobre cada uno de los elementos obtenidos de la consulta. El fichero `recordFunctions.js` define las funciones utilizadas como argumento de `runOne` y `runMultiple`. Se han definido dos funciones en este fichero, la primera de ellas `formatGeneStartEnd` es utilizada en la consulta `/by_pos` de la API. Dado un gen los transforma en un objeto con dos atributos: posición de inicio y posicón de fin.

Como hemos visto en el fichero `variations.js` en la página 93, las consultas realizadas a la base de datos obtienen la variación, sus variedades y el gen al que afectan. La segunda función, `formatVariationDataTable`, proporciona información adicional añadiendo las anotaciones funcionales relacionadas con el gen. Para ello, hace uso del diccionario global obtenido en el proceso de instanciación del servidor. De este modo se mantiene un equilibrio entre la complejidad de las consultas y la rapidez de las mismas (Fragmento de código A.21).

El directorio security (Figura 3.29) consta de dos ficheros. El primero de ellos, exporta un *middleware* para ser utilizado por *express* con el objetivo de filtrar las peticiones según la ip del cliente. En el fichero de configuración se define una expresión regular que deben cumplir las direcciones ip para que se les permita acceder al Servicio. La función definida comprueba si el cliente está autorizado. Si lo está, el cliente podrá acceder a los distintos servicios del servidor. En caso contrario, recibirá un mensaje de error indicándosele que no está autorizado a acceder a los servicios del servidor.

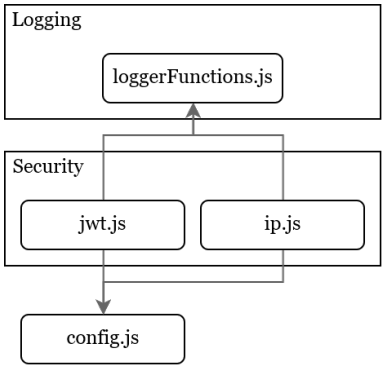

Figura 3.29: Diagrama de componentes del paquete security con sus dependencias

El segundo gestiona la comunicación entre *frontend* y *backend* mediante el uso de *tokens*. El cliente debe incluir un *token* válido que el servidor comprobará en el servicio que ofrece *Auth0*. La tecnología empleada es *JSON Web Tokens (JWT)*. Cuando el servidor recibe una petición, comprueba la validez del token mediante la librería `express-jwt`.

La carpeta public almacena el contenido que se enviará al cliente (aplicación generada con el *framework* Angular, para más detalles véase Sección 3.5).

En el paquete routes (Figura 3.30) se almacenan los distintos ficheros que contienen las funciones a ejecutar cuando se realizan peticiones a la API. En la Figura 3.30 se muestra sólo uno de los seis ficheros existentes ya que las dependencias son las mismas para todos los ficheros.

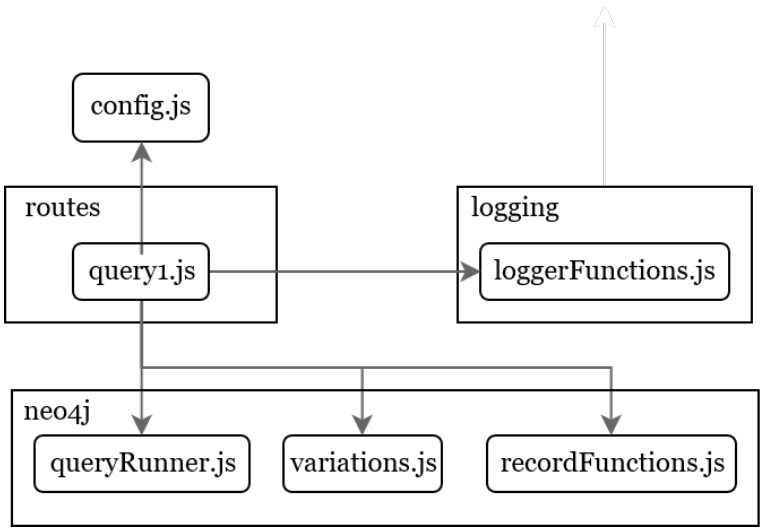

Figura 3.30: Diagrama parcial de componentes del paquete routes con sus dependencias

Los seis ficheros siguen el mismo patrón de ejecución de la Figura 3.31. En primer lugar, obtiene los parámetros con los que se creará la consulta del cuerpo de la petición. En segundo lugar genera la consulta llamando a la función correspondiente del fichero `variations.js` del paquete neo4j. Una vez ha obtenido la consulta la ejecuta llamando al fichero `queryRunner.js` del paquete neo4j pasándole la función para procesar los registros del fichero `record_functions.js` del paquete neo4j. Cuando ha recibido el resultado de la consulta almacena en el log correspondiente la información y envía la respuesta al cliente.

3.5 Frontend

En esta sección se muestra la aplicación web para que el usuario interaccione con el sistema. Para su implementación se ha empleado el *framework* de desarrollo de aplicaciones Angular[28] así como Angular-CLI [29] para hacer más eficiente el flujo de trabajo. Se ha hecho uso de Angular Material[30]. Consta de una serie de componentes predefinidos que ayudan a crear una interfaz consistente y optimizada.

A continuación, se detallan algunos conceptos importantes para tener un entendimiento claro de cómo funciona Angular:

– Directiva: modifica la apariencia o comportamiento del Modelo de Objetos del Documento (DOM) de un elemento. Existen tres tipos de directiva: componentes (directivas con una plantilla HTML), directivas estructurales (modifica el DOM añadiendo o eliminando elementos al mismo) y directivas de atributos (cambia la apariencia o comportamiento de un elemento, componente o directiva).

[28] https://angular.io/
[29] https://cli.angular.io/
[30] https://material.angular.io/

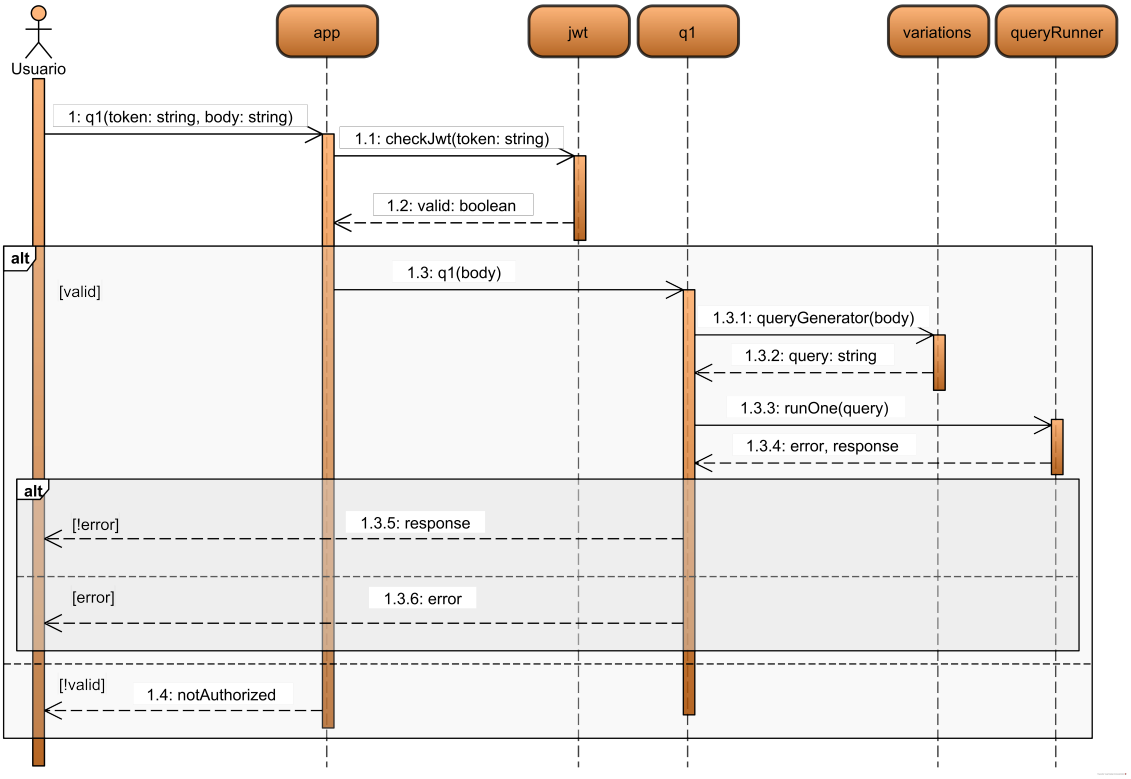

Figura 3.31: Diagrama de secuencia de los ficheros del paquete routes

– *Router*: servicio que se encarga de mostrar la vista de un componente dada una url. Es necesario configurar las rutas que va a gestionar el *router*. Una vez se ha proporcionado la configuración, las vistas se mostrarán donde se haya definido la directiva de `router-outlet`.

– Decorador: permite añadir metadatos extra a un componente.

Para el desarrollo de este prototipo de aplicación de usuario se han generado diez componentes localizados dentro de un único módulo. En Angular, las aplicaciones se construyen a partir de módulos. El sistema de modularización de Angular se llama *NgModules*. Puede contener componentes, proveedores de servicios y código que esté dentro del ámbito del módulo. Este sistema permite importar funcionalidad de otros módulos y, a su vez, exportar parte de su funcionalidad.

Un módulo se define como una clase con el decorador `@NgModule`. Este decorador recibe un objeto con múltiples propiedades, a saber:

– *declarations*: conjunto de componentes, directivas, etc. pertenecientes al módulo.

– *exports*: subconjunto de declaraciones que serán visibles y accesibles desde el exterior.

– *imports*: módulos cuyas clases exportadas son utilizadas por el módulo.

– *entryComponents*: conjunto de módulos cargados de manera imperativa

– *providers*: conjunto de servicios accesibles por la aplicación.

– *bootstrap*: principal pista de la aplicación, que contiene el resto de vistas de la aplicación.

Por tora parte, un componente es una clase con el decorador `@Component` que controla una parte de la interfaz de usuario, llamada vista, mediante la lógica definida dentro de esta. Estos componentes son creados, actualizados y destruidos según el usuario navega por la aplicación. Este decorador presenta las siguientes opciones de configuración:

– *selector*: selector CSS utilizado para generar instancias del componente en las plantillas HTML.

– *templateUrl*: plantilla HTML del componente. En otras palabras, es la vista que gestiona en componente.

– *providers*: servicios inyectados en el componente.

La plantilla de los módulos está formada por HTML pero con características extra que permiten modificar la plantilla en función de la lógica de la plantilla. La plantilla puede sincronizar atributos de la plantilla con atributos de la aplicación, modificar de manera dinámica la información antes de mostrarla o modificar la estructura de la plantilla en función de la lógica de la aplicación.

Componentes

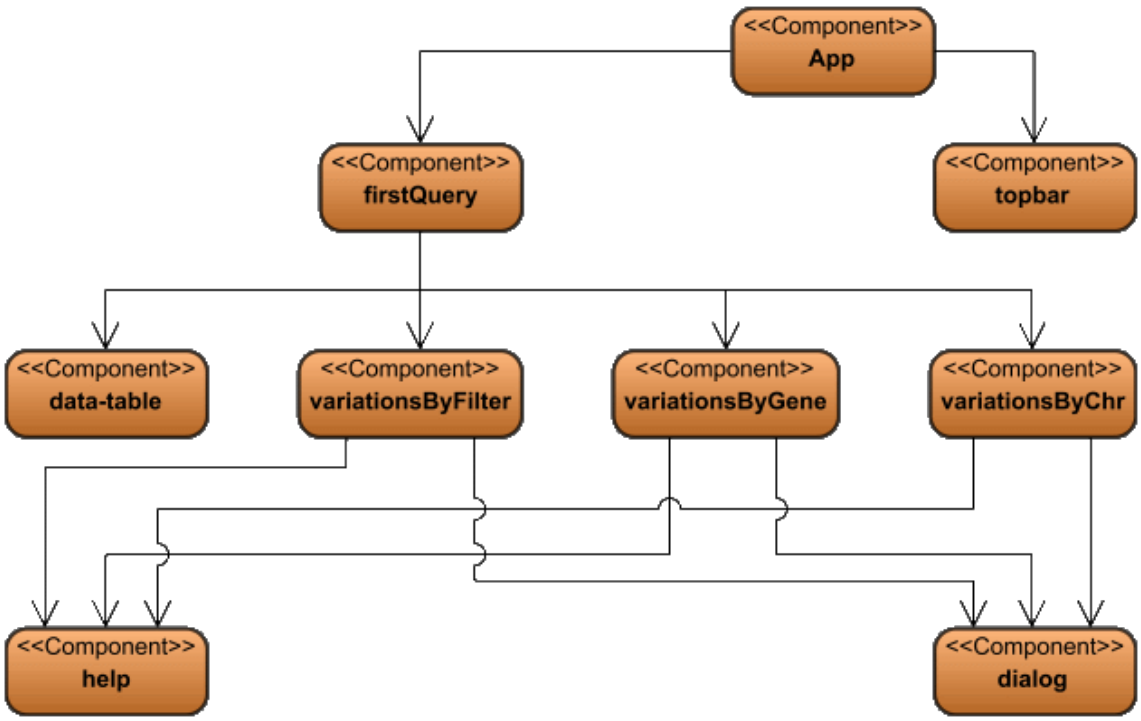

Figura 3.32: Diagrama de componentes de la aplicación web

El primero de ellos es el componente principal de la aplicación y punto de entrada (*AppComponent*). Únicamente contiene el componente de la barra superior

(que gestiona el redireccionamiento del *router*) y una directiva llamada *router-outlet*, que actúa como un contenedor que carga dinámicamente los componentes en función del estado del *router*.

El segundo componente, llamado *topbarComponent* se encarga de la redirección del *router* de la aplicación, así como gestionar la autenticación: gestiona el inicio y cierre de sesión. En la Figura 3.33 podemos observar el estado del componente *topbarComponent* al entrar en la aplicación y una vez se ha iniciado sesión.

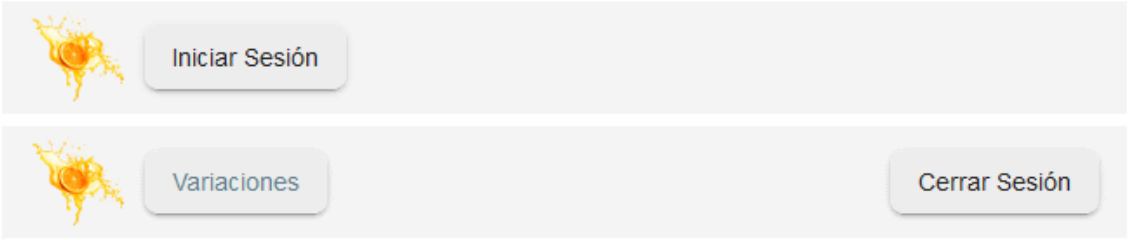

Figura 3.33: Componente topbarComponent antes y después de iniciar sesión

Al intentar iniciar sesión se nos redireccionará al servicio encargado de contactar con *Auth0* para validar la información (más información en la página 102). Esta nueva vista puede ser observada en la Figura 3.34.

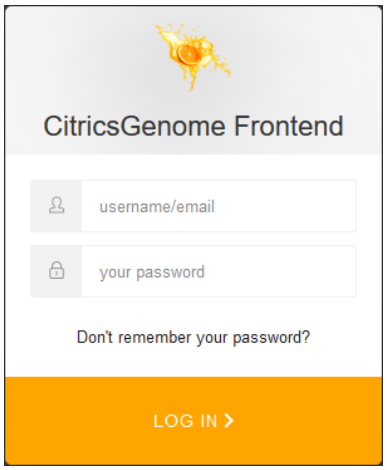

Figura 3.34: Inicio de sesión a través de Auth0

Si el inicio de sesión se ha realizado correctamente, el componente *topbarComponent* muestra nuevas opciones que no estaban disponibles antes de iniciar sesión.

A continuación, se presenta un componente auxiliar utilizado en la validación de los formularios. Se trata de un diálogo que será instanciado si la validación de los campos de los formularios no es la correcta (DialogComponent). Debido al uso que se le da, es necesario incluir a este componente en la lista de "entryComponents". El componente es inyectado en los componentes encargados de presentar los filtros. Cuando se crea un nuevo diálogo, el mensaje de error es enviado como parámetro a su constructor, permitiendo reutilizar el componente para cualquier tipo de error.

El componente llamado *FirstQueryComponent* se encarga de realizar todas aquellas consultas cuyo objetivo sea la obtención de variaciones, ya que el formato de

respuesta será el mismo (definido en la página 103). Por tanto, contiene: un componente por cada tipo de consulta sobre variaciones y un componente encargado de mostrar en formato tabla las variaciones obtenidas de las consultas. Una vez se ha completado una búsqueda, mostrará un botón para permitir descargar en formato CSV la tabla obtenida.

El componente *VariationsByFilterComponent* se encarga de obtener variaciones utilizando la ruta `/q1` de la API. En la Imagen 3.35 se puede observar la interfaz definida para la selección de los filtros. Además, al igual que el resto de componentes cuyo fin es la obtención de variaciones mediante la utilización de filtros, contiene una pestaña de ayuda, con una instancia del componente `help`.

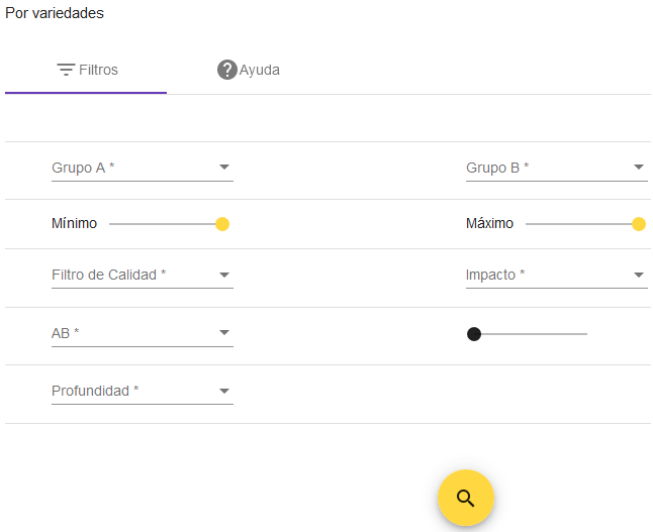

Figura 3.35: Filtros consulta por variedades

El componente *VariationsByGenesComponent* se encarga de obtener variaciones utilizando la ruta `/by_pos` de la API. En la Imagen 3.36 se puede observar la interfaz definida para la selección de los filtros.

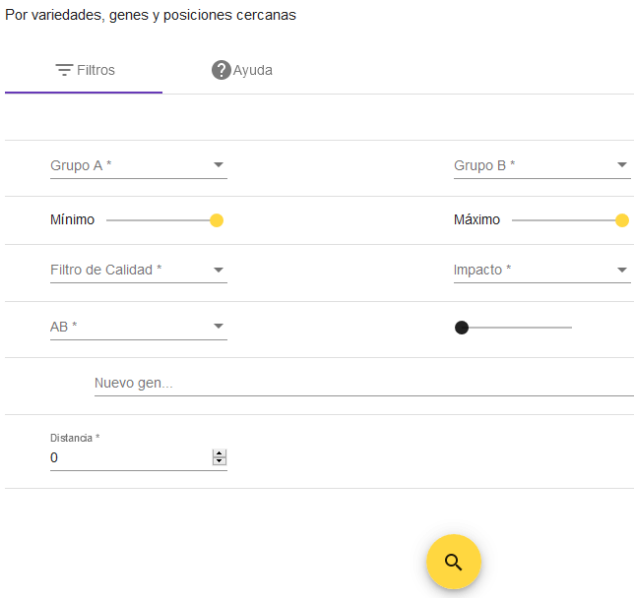

Figura 3.36: Filtros consulta por genes

El componente *VariationsByChrComponent* se encarga de obtener variaciones utilizando la ruta /by_pos de la API. En la Imagen 3.37 se puede observar la interfaz definida para la selección de los filtros.

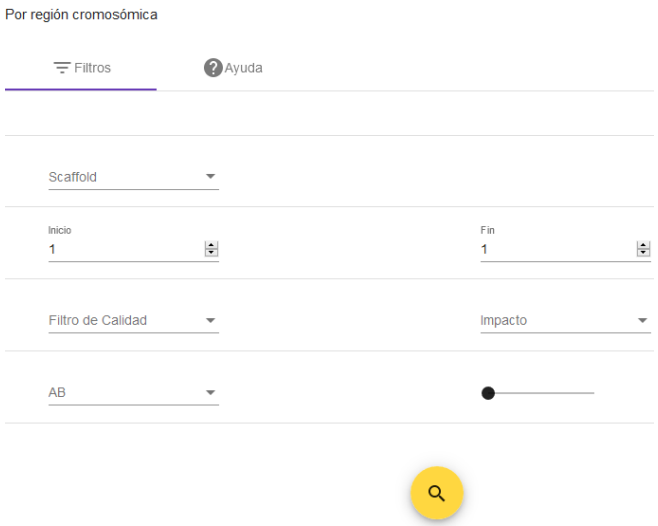

Figura 3.37: Filtros consulta por cromosomas

El último cponente definido (*HelpComponent*), recibe como parámetros en su construcción la descripción de una consulta y los pasos que se debenrealizar para rellenar el formulario. A partir de estos datos, define en su plantilla un acordeón con dos elementos, la descripción de la consulta y los pasos a seguir mediante la implementación de un *stepper* (componente definido en la librería Angular Material).

El componente *dataTableComponent* contiene la tabla donde se presentarán los resultados obtenidos de las consultas. Los atributos que la tabla muestra son:

- tipo, *scaffold*, alelo de referencia y alelo alternativo de la variación así como el valor del atributo *Allele Balance* de los nodos lectura pertenecientes a la variación.
- anotación, el tipo de efecto que provoca en el gen al que afecta.
- gen al que afecta la variación.
- anotaciones funcionales relacionados con el gen, a saber: GOs, dominios, enzimas y *pathways*.
- variedades en las que aparece la variación.

La tabla puede ser ordenada por cualquiera de sus columnas y descargarse en formato csv para trabajar con los datos en otras herramientas.

Servicios

En Angular, el concepto de servicio engloba cualquier valor o dato que necesita una aplicación. Se le supone un único y definido propósito. La existencia del

concepto de servicio responde a motivos de reusabilidad y modularidad. Mientras que un componente contiene la lógica relacionada con su plantilla HTML, no debería contener lógica ajena a su plantilla. Por ejemplo, un componente no debería lanzar peticiones a un servidor externo. Para la lógica no relacionada con la interacción entre la interfaz y el usuario el componente puede delegar estas tareas y utiliza servicios.

En un servicio se utiliza el decorador `@Injectable`. Con este decorador, se proporciona a Angular los metadatos necesarios para inyectar el servicio como una dependencia. Esta inyección se realiza en el constructor del componente. Se han creado cinco servicios en el prototipo.

El servicio `Auth0Service` se encarga de gestionar la autenticación de usuarios, es el servicio utilizado por el componente topbarComponent. Proporciona métodos para iniciar sesión en base a un nombre de usuario y contraseña, cerrar sesión y comprobar si se ha iniciado sesión.

El servicio `Auth0GuardService` implementa la interfaz `CanActivate` de Angular. Cuando se intenta acceder a alguna página de la aplicación que no sea la principal, se activará esta guarda que comprueba si se ha iniciado sesión. En caso afirmativo se proporciona acceso a la página, si no, se redirecciona a la página principal.

El servicio `FiltersService` proporciona los datos utilizados para rellenar los formularios. Se diferencian dos tipos de datos: estáticos y dinámicos. Los estáticos son aquellos cuyo contenido no varía en la base de datos:

- variedades: lista con todas las variedades presentes en la base de datos.
- impacto de anotación: tipos del valor del atributo *annotation_impact* de los nodos de tipo `annotation`. Sus posibles valores son *HIGH*, *MODIFIER*, *MODERATE*, *LOW*
- profundidad: posibles valores de la profundidad posicional de las variaciones. Sus posibles valores son: todas, génicas, no génicas, en exón, en intrón, en UTR y en CDS.
- operaciones sobre campo AB: mayor que, mayor o igual que, menor que y menor o igual que.
- *scaffolds*: lista completa de los *scaffolds* presentes en la base de datos.

En cuanto a los datos dinámicos están los filtros sobre los campos DP y GQ definidos por el usuario. Puesto que esto sí puede cambiar es necesario consultar al servidor por si se ha modificado. Se consulta dicha información a la ruta /fc de la API.

El servicio `TokenService` obtiene un *token* para poder comunicarse con el *backend*. A esta comunicación se la conoce como comunicación de máquina a máquina (M2M). Para evitar que cualquiera pueda acceder a la API del *backend* se comprueba que las peticiones tengan un token válido antes de procesar la respuesta. La obtención del *token* está gestionada por *Auth0*.

El servicio *VarietiesService* se encarga de realizar consultas a la base de Neo4j y obtener el resultado. Recibe dos parámetros (Fragmento de código A.22). El

primero de ellos, `path`, indica la ruta a la cual se va a realizar la consulta; el segundo, `body` contiene el cuerpo de la petición: los valores a utilizar para generar la consulta en el servidor.

Pipes

Un *pipe* permite realizar transformaciones en los datos que se van a mostrar en la aplicación web. Toma un conjunto de datos y produce como salida ese mismo conjunto de datos transformados de la manera deseada. Estos *pipes* pueden utilizarse dentro de las plantillas HTML y Angular realizará la transformación de manera transparente.

Se ha creado un *pipe* (*FilterPipe*) para los valores a elegir del campo filtro de los formularios. Recibe como entrada datos del tipo "{numero_1}-{numero_2}" y los transforma en datos del tipo "DP: {numero_1}, GQ: {numero_2}".

Interfaces

El uso de interfaces proviene del lenguaje TypeScript, utilizado por Angular. Una interfaz es una entidad cuyo fin es identificar los tipos de datos con los que se trabaja. De este modo se obtiene una programación más fiable y segura a costa de sacrificar flexibilidad. En la aplicación web se ha definido una interfaz auxiliar y dos interfaces generales: una para el formato de los datos de la tabla y otro para el formato de la respuesta de la consulta implementada. Con respecto a la primera (Q1DataTable) contiene los tipos de las trece columnas de la tabla. Con respecto a la segunda (Q1Response), presenta el formato de las respuestas del servidor: el campo `count` con el total de variaciones y el campo `variations`, un array de elementos definidos en la interfaz auxiliar `Q1ResponseTable`, cuyo formato puede ser consultado en el Fragmento de código A.23 .

CAPÍTULO 4
Validación de la Solución

Existen dos formas de evaluar de manera empírica una tecnología: validación y evaluación [50]. La validación se define como la evaluación de una simulación de dicha tecnología en uso en un contexto de interés con el objetivo de predecir qué sucedería si se utilizara en el mundo real. Por otro lado, la evaluación se entiende como la evaluación empírica de una tecnología cuando se utiliza en condiciones reales.

En este capítulo se incluyen todas las reuniones realizadas con los *stakeholders*. Estas reuniones sirven como hilo conductor de la tesis. En ellas puede verse el avance del proyecto y en qué puntos han ido surgiendo los conceptos e ideas más importantes e influyentes.

Existen diversos métodos de validación empírica:

- Opinión experta: obtención de la opinión de expertos mediante el uso de entrevistas, cuestionario o grupos de enfoque.
- Experimento de caso único: probar un prototipo con un ejemplo caso de estudio. El caso puede ser ideal o real y realizarse en el laboratorio o en condiciones reales.
- Investigación técnica: Uso de un prototipo por los *stakeholders*.
- Experimentos estadísticos: comparación de los efectos del uso de dos o más prototipos en múltiples muestras de población o contextos de uso.

4.1 Diseño de la Validación

Para la validación de la solución se han empleados un método: opinión experta.

Opinión experta

Se realiza durante la validación conceptual de la solución, antes de que se valide el artefacto. Los investigadores pueden consultar la opinión de los expertos

en cuestiones referentes a la usabilidad y utilidad de dicho artefacto. No se trata de una encuesta estadística para obtener una distribución de las opiniones de toda la población de expertos, sino que es un intento de extraer información temprana de las expectativas que se tienen del artefacto en un contexto real, fuera del laboratorio. Las opiniones positivas o acríticas carecen de valor, sin embargo, opiniones negativas o críticas permiten errores y asegurar que se cumplen las expectativas de los expertos mediante la obtención de información útil. En primer lugar, se debe analizar el tipo de perfil de los expertos que nos van a proporcionar la información. Se deben obtener los objetivos de la solución partiendo de sus necesidades. Estos objetivos surgen como un intento de mejora de las necesidades obtenidas.

Antes y durante el proceso de desarrollo de la solución se han realizado múltiples reuniones con los expertos. Estas reuniones se han realizado con el objetivo de tener el mayor entendimiento posible del dominio y asegurar que el progreso de la solución es el adecuado. Se han definido de manera temprana la usabilidad y funcionalidad de la solución. Además, se realizó una sesión de trabajo práctica con algunos expertos para poder determinar e identificar qué interacciones deseaban tener a su alcance en la aplicación. Esta sesión de trabajo práctica fue especialmente útil para diseñar las interfaces de usuario pues obligó a los expertos a plantearse de manera concreta lo que esperaban obtener y con qué elementos deseaban interactuar al utilizar la aplicación. Hasta el momento, se han realizado trece reuniones y dos sesiones de trabajo interactivas para obtener toda la retroalimentación posible antes y durante el proceso de desarrollo de la herramienta.

4.2 Ejecución de la Validación

Opinión experta

Con el objetivo de obtener toda la información y retroalimentación posible se han realizado varias reuniones presenciales, una sesión de trabajo práctico y una reunión no presencial. A continuación, se detallan los cambios y decisiones tomadas en cada una de estas reuniones.

En la primera reunión se realizó una primera toma de contacto. Los expertos del IVIA mostraron la base de datos proporcionada por el Centro de Investigación Príncipe Felipe[1]. Esta base de datos es utilizada para descargarse secuencias genómicas que posteriormente son cargadas en Excel. Descargan secuencias de referencia de distintas especies y las comparan con las variaciones que han secuenciado ellos. Cada secuencia viene desglosada jerárquicamente de elementos más generales a más concretos (desde genes hasta CDS). Además de las variaciones y las secuencias, trabajan con proteínas, *pathways*, enzimas, notaciones funcionales obtenidas con la herramienta *Blast-To-GO* (GO), etc. Nos comunican que deseanconocer cómo cambian los genes en las distintas variedades y como afectan al resto de elementos.

De esta reunión se tuvo una primera toma de contacto con el dominio a estudiar, proporcionando información básica para comenzar a trabajar en el mo-

[1] http://www.cipf.es/

delado conceptual. Finalmente, los expertos del IVIA se comprometieron a proporcionarnos todos los ficheros originales con los que ellos trabajan para poder estudiarlos meticulosamente y la herramienta para anotar las variaciones: SnpEff.

La segunda reunión se centró en la resolución de dudas surgidas a partir del estudio de los ficheros proporcionados por el IVIA. Las relaciones de parentesco entre los elementos génicos tienen una mayor complejidad de la esperada. Debe tenerse cuidado con el proceso corte y empalme donde se pueden producir distintos mRNA con el mismo identificador. Por cada fichero se ha realizado un repaso exhaustivo de cada uno de los elementos y atributos que aparecen. Además, nos informan de cómo desean interrogar los datos: buscando genes siempre y a partir de ahí expandir la búsqueda a sus anotaciones funcionales.

De esta reunión se obtuvo un gran conocimiento del formato y contenido de los ficheros proporcionados, así como un primer acercamiento a la forma en que desean explotar los datos.

La tercera reunión fue especialmente ágil, con una primera versión de la base de dato, cargando parcialmente los datos de mayor interés un unas pocas variedades, se realizaron consultas junto con los expertos del IVIA. Esta reunión nos permitió obtener información sobre cómo deberían relacionarse los elementos y generar nuevas relaciones que no se habían tenido en cuenta.

Respecto a la cuarta y la quinta reunión se siguió la dinámica de la tercera, pero incrementando la complejidad de las preguntas. En la reunión anterior las consultas se limitaron a localizar y comparar variaciones y relacionarlas con los genes que modifican. En estas reuniones se añadieron dos nuevos nodos a las consultas: enzimas y *pathways*. De estas reuniones, como de la anterior, obtuvimos conocimiento extra sobre qué tipo de consultas desean realizar y los caminos apropiados a seguir dentro del grafo para interrogar los datos.

En la sexta reunión se trataron varios temas importantes: el primero de ellos es sobre el uso de la herramienta SnpEff. Al anotar las variaciones se producen varios errores referentes a la identificación de proteínas y CDS. El IVIA propone una forma alternativa de identificación de estos elementos con el objetivo de eliminar estos problemas. El segundo de ellos tiene que ver con el filtrado de los datos, el IVIA indica que necesita gestionar los filtros de calidad de manera dinámica ya que es posible tener que realizar la misma consulta múltiples veces con distintos parámetros de calidad. Se proporciona una primera versión del filtro de calidad con tres atributos a tener en cuenta DP, GQ y AB. Se proponen dos formas de aplicación del filtro de calidad: bien realizar el filtrado antes de crear la base de datos y generarla con un subconjunto de las variaciones o incorporar el filtro en las consultas. Tal y como se ha indicado anteriormente, el IVIA necesita tener un control dinámico de los filtros, por lo que la primera opción queda descartada. Por otra parte, se establece cómo debe identificarse una variación: por posición, cromosoma, alelo de referencia y alelo alternativo.

Se han generado un conjunto de seis preguntas sobre las que ir trabajando:

1. ¿Cuántas variaciones hay en A y no en B siendo A y B conjuntos de variedades? Esta consulta se refiere a entradas en los ficheros VCF, identificando las variaciones como nos ha indicado el IVIA

2. ¿Cuántas variaciones hay en B y no en A siendo A y B conjuntos de variedades? Esta consulta puede responderse ejecutando de forma inversa la consulta anterior.

3. Dadas dos grupos de variedades, A y B, obtener los *pathways* en los que interviene una enzima codificada por un mRNA que está afectado por una variación presente en A y no en B

4. Obtener *pathways* tales que interviene una enzima que está codificada por un mRNA afectado por una variación con una anotación cuyo impacto es de tipo "HIGH". Es el mismo tipo de consulta que la anterior, pero añadiendo un criterio de búsqueda más.

5. Obtener, para cada *pathway*, las variedades que poseen variaciones con un impacto "HIGH" que afectan a mRNAs que codifican enzimas que intervienen en dicho *pathway*

6. Obtener *pathways* con enzimas codificadas por mRNAs afectados por variaciones presentes en un grupo de variedades A que no están en el grupo de variedades B.

El IVIA define los atributos que desea obtener de cada variación: gen, posición, alelos, cromosoma, *pathway* al que afecta la variación, enzima a la que afecta la variación, el impacto y el tipo de efecto (tipo de anotación).

En la séptima reunión se solucionaron los problemas restantes en el uso de la herramienta SnpEff. Respecto a los filtros de calidad se materializó un procedimiento eficiente para hacer uso de ellos y se mostraron las consultas definidas para las preguntas planteadas en la reunión anterior. Las consultas se han agrupado en cuatro eliminando redundancias.

En la octava reunión se presentó el modelo completo de la base de datos a los expertos del IVIA para discutirlo. Se acuerda cargar el total de las variedades ya que las pruebas realizadas con cuatro han resultado satisfactorias hasta el momento. Se establece la necesidad de realizar una sesión práctica para determinar mejor los requerimientos de las preguntas a implementar en el prototipo. IVIA solicitó la adición de más elementos referentes a proteínas: GO y dominios.

En la novena reunión se ejecutaron las consultas con la base de datos completa: con las cuarenta variedades cargadas. Se han detectado problemas de eficiencia que hará necesario refinar las consultas actuales y cualquier nueva consulta que solicite el IVIA. Pos su parte, el IVIA comenta que para ellos la generación de nuevas consultas no es trivial a nivel conceptual y les puede llevar un período de tiempo considerable. Se han definido nuevos filtros de calidad para ser implementados en la base de datos.

La décima reunión coincidió con la sesión práctica. En la reunión, se han realizado diversas consultas en lo referente a las enzimas y proteínas y la relación entre ellas. Se concluye que debe existir una relación entre la enzima y la proteína. No obstante, no se aclara cómo obtener esa relación a partir de los ficheros proporcionados ni si está presente. Aparentemente, la aplicación del filtro de calidad debe realizarse después de las uniones e intersecciones entre las variedades. Esto es debido a que, tras realizar la prueba, se proporciona un resultado

aparentemente más fiable, aunque requiere de mayor investigación. Además, Se realizarán algunas propuestas en base a parámetros de entrada proporcionados por el IVIA para comprobar la validez de los resultados

De esta reunión han surgido modificaciones en el Esquema Conceptual de los Cítricos para adaptarse a las nuevas relaciones. No sólo entre enzima y proteína, sino entre proteína y dominio y GO.

En la sesión práctica se pidió a los investigadores que participaran en el proceso de creación de la interfaz de usuario. Durante la sesión, los investigadores crearon desde cero un prototipo de interfaz, guiados por los informáticos, que iban extrayendo conceptos y operaciones mediante preguntas. Esta sesión ha servido para generar la interfaz de usuario del prototipo. Con esta interfaz de usuario, las consultas a implementar se reducen a dos, ya que con ellas y la interfaz de usuario se pueden responder a las cuatro que se habían definido. Tras esta reunión se generaron las preguntas de investigación utilizadas en el desarrollo de esta tesis.

En la undécima reunión, se analizaron los resultados obtenidos de ejecutar las consultas propuestas. Se observa que para algunos parámetros no se obtienen los resultados esperados. Tras un estudio de los mismos se llega a la conclusión de que esto es debido a los diferentes filtros de calidad utilizados para las consultas. La existencia de falsos positivos y falsos positivos hace replantearse si la configuración actual de los filtros es la correcta. Se deduce que es necesario seguir experimentando con diferentes configuraciones.

Se ha mostrado el prototipo desarrollado, ejecutándose en el momento consultas y analizando los resultados. Como resultado de estas acciones se concluye que la información aportada por el prototipo es de utilidad, aunque es necesario añadir funcionalidad extra en la selección de filtros, la flexibilidad de las consultas y la profundidad de los elementos afectados. Se plantea un nuevo tipo de consultas que siga un patrón inverso. En estas, en lugar de iniciar la búsqueda partiendo de las variaciones, se utilizará un rango posicional o se partirá de la anotación funcional de algún gen (proteína, etc.).

Posterior a esta reunión se acordó modificar el filtro de calidad de forma que tan sólo utilizara los atributos DP y GQ, extrayendo el atributo AB, que antes formaba parte del filtro de calidad, a una cláusula más dentro de la consulta. También se ha reafirmado la necesidad de permitir una cierta flexibilidad a la hora de permitir que se tengan en cuenta variaciones que aparezcan en un porcentaje del total de variedades definidas. Finalmente, se han especificado los requisitos para la generación de la nueva consulta, que buscará variaciones partiendo de genes

En la doceava reunión se produjo un cambio sustancial en el conjunto de consultas definidas. De las consultas definidas en la sexta reunión, las consultas 3, 4, 5 y 6 fueron eliminadas de los objetivos a corto plazo. En su lugar se añadieron dos nuevas consultas. La primera de ellas consiste en buscar variaciones que se encuentran posicionalmente dentro de un conjunto de genes definidos por el usuario. Sobre estas consultas se aplicarán los filtros de calidad de las consultas 1 y 2. La segunda consulta consiste en buscar variaciones que se encuentren posicionalmente en regiones cromosómicas definidas. En esta última consulta no se tendrán en cuenta grupos de variedades ni filtros posicionales, pero sí filtros

de calidad. De esta reunión se acordó realizar las modificaciones pertinentes y presentar la nueva versión de la aplicación.

En la decimotercera y última reunión se presentó la aplicación realizando varias pruebas para comprobar la fiabilidad de los datos. Dándose resultados no satisfactorios se analizó *in situ* las posibles causas. Se determinó que la causa era la herramienta SnpEff. Por defecto, el valor ud (*upstream/downstream*) es de 5000. Este valor era demasiado alto y generaba falsos positivos. Se concluyó que se debía regenerar la base de datos utilizando un valor de 1000 y volver a realizar las pruebas.

4.3 Análisis de resultados

Se identifican dos puntos relevantes a la hora de analizar los resultados. Por un lado, el aprovechamiento de la herramienta y su utilidad para los usuarios finales. Para ello se han realizado demostraciones en las reuniones, así como ejemplos concretos en entornos de prueba controlados. La realización de estas demostraciones arroja unos resultados positivos ya que permiten a los usuarios trabajar con un volumen de datos inabordables anteriormente. Anteriormente se limitaban a analizar manualmente pequeñas regiones del genoma que sabían que podían ser importantes e intentar compararlas utilizando un limitado número de variedades, mientras que en las pruebas ejecutadas se ha mostrado posible:

- Comparar grandes regiones del genoma. Donde anteriormente se debía limitar a pequeñas regiones con un alto interés. Con la herramienta esta limitación se ha eliminado y es posible analizar, por ejemplo, genes enteros o incluso regiones cromosómicas de longitud arbitraria dentro de unos parámetros de tiempo reducidos.

- Comparar grupos de variedades más numerosos. Donde anteriormente las comparaciones entre variedades estaban limitadas, ahora se pueden comparar grupos mucho más numerosos sin que repercuta negativamente en el rendimiento.

Por otro lado, el aumento de conocimiento en el área estudiada. Durante las reuniones y, especialmente, para la generación del esquema conceptual, se produjo un meticuloso estudio de los datos. De este estudio surgieron dudas que ayudaron a incrementar el conocimiento de los propios datos, así como su interpretación por parte del IVIA. Ejemplo de ello pueden ser la definición de *scaffold*, que inicialmente no estaba del todo clara o el valor adecuado del campo ud para la herramienta SnpEff y sus efectos en los datos. También se detectaron problemas de calidad en los datos de entrada como valores anómalos, repetidos o faltantes.

Así pues, se concluye que la herramienta ha sido provechosa tanto durante el desarrollo de la misma como en las demostraciones de uso y se espera que pueda beneficiar todavía más a los usuarios finales una vez de despliegue en un contexto real.

CAPÍTULO 5
Conclusiones

Esta tesis se originó con un propósito bien claro y definido, el de mejorar las herramientas de análisis de datos genómicos actuales para aumentar su eficiencia y rapidez mediante la creación de una nueva, con capacidad para el tratamiento masivo de datos, escalabilidad, simplicidad de uso y eficiencia en el proceso de análisis. De este propósito surgieron un conjunto de preguntas de investigación que han sido respondidas a lo largo de este documento. Respecto a la primera, y única pregunta de conocimiento, pregunta de investigación: ¿Qué características definen el tipo de datos involucrados en el ámbito genómico-cítrico y cómo trabajan con ellos en el IVIA? se puede afirmar que ha sido respondida, por un lado, en la Sección 3.1 donde se caracterizan y detallan los datos con los que se trabaja en este ámbito y, por otro lado, en la Sección 4.2 donde el IVIA, durante las reuniones realizadas, ha expuesto de manera minuciosa qué tratamiento le dan a los datos, cómo trabajan y qué desearían obtener de ellos.

En cuanto a la segunda pregunta de investigación, y primera de diseño: Diseño de un esquema conceptual del genoma de los cítricos, se ha producido el artefacto deseado, un esquema conceptual del genoma de los cítricos. Este esquema, detallado en la Sección 3.1, ha permitido entender el dominio del problema, sus limitaciones y necesidades. Todo esto ha generado una sólida base para continuar con el trabajo.

Referente a la última pregunta de investigación, y segunda de diseño: Diseño de una aplicación que mejore el proceso de análisis genético por el IVIA en el ámbito genómico-cítrico, se ha generado una aplicación web para mejorar dicho proceso. Tomando como base el ECGCit., se ha diseñado e implementado una aplicación web que cubre las necesidades actuales del IVIA. Durante la segunda parte de este documento, especialmente el Capítulo 3, se ha mostrado el proceso de implementación de la aplicación.

La complejidad del dominio y las exigencias del caso de uso han resultado en un entendimiento profundo de los datos, los procesos de trabajo y las metas del IVIA. Especialmente durante el proceso ETL en el cual se ha puesto un especial esfuerzo para entender el dominio y generar una representación lo más real y completa posible primando la eficiencia. Esto ha redundado en un mayor conocimiento del dominio genético y genómico.

A nivel técnico el trabajo ha servido para explorar nuevas tecnologías como por ejemplo el lenguaje de programación Python, la programación multihilo, el

framework Angular, bases de datos orientadas a grafos, nociones de redes y seguridad o administración de sistemas. El área de mayor investigación ha sido la tecnología de bases de datos orientadas a grafos. A través de un exhaustivo estudio comparativo se ha determinado como la mejor opción para las características de los datos con los que se trabajaba. De este modo, se ha dedicado un tiempo considerable en el diseño e implementación de la base de datos con el objetivo de optimizar el sistema al máximo.

Los artefactos generados son útiles para la mejora de procesos para el IVIA tal y como se observa en la Sección 4.2. El ECGCit. ha sacado a la luz diferentes deficiencias: el tratamiento de los datos era incorrecto en algunos casos, encontrándose inconsistencias; la existencia de conceptos erróneamente definidos que generaban confusión y ralentizaban el flujo de trabajo. Otro punto destacado es el erróneo uso de la herramienta SnpEff ya que de no ser por el uso de la aplicación web no se hubiera detectado dicho uso erróneo. Por otro lado, se ha comprobado experimentalmente que los tiempos necesarios para ejecutar análisis se han visto reducidos drásticamente.

La plataforma desarrollada cumple con los cuatro objetivos exigidos:

- Capacidad de almacenamiento masivo: se ha almacenado el conjunto total de información que se poseía. Este almacenamiento se ha realizado tras un proceso de análisis y estructuración de los datos asegurando que no sólo se han almacenado, sino que se han almacenado correctamente.

- Escalabilidad: debido a la naturaleza de JavaScript, la escalabilidad se logra lanzando instancias de los procesos para repartir las peticiones. La aplicación consumo en todo momento menos de 50 MB de memoria RAM y un porcentaje de uso de la CPU menor del 4 %. Con estos datos se garantiza poder lanzar un alto número de procesos en función de la carga.

- Simplicidad de uso: se ha comprobado experimentalmente que la interacción con la aplicación es clara y directa, proporcionando una experiencia de uso satisfactoria.

- Eficiencia en el proceso de análisis: se ha comprobado experimentalmente una mejora en la eficiencia del proceso de análisis. Esta mejora ha sido posible gracias al proceso de análisis y estructuración de los datos así como el diseño eficiente de la herramienta. Gracias a la plataforma ahora es posible realizar en períodos de tiempo razonables consultas y análisis que antes simplemente no eran abarcables.

5.1 Trabajo futuro

Con el desarrollo de este TFM se finaliza la primera versión del prototipo. A partir de este punto se inicia la iteración para generar la segunda versión del prototipo. Los principales objetivos a corto plazo son incluir las consultas referidas a la obtención de variaciones partiendo de anotaciones funcionales. Estas consultas, si bien estaban presentes en las primeras reuniones, se decidieron posponer

en favor de otras más prioritarias. A medio plazo, se tiene en mente aumentar el tipo de consultas a realizar a la base de datos.

El ECGCit. no es una entidad cerrada. Está abierto a modificaciones y actualizaciones según aumente el conocimiento del dominio y varíen las necesidades de los *stakeholders*. De este primer esquema se está estudiando la posibilidad de generalizar el proceso de creación de aplicaciones de este tipo para realizar análisis similares en otros cultivos como los cereales. A más largo plazo está planeada la unificación de este esquema con el Esquema Conceptual del Genoma Humano (ECGH) [51]. El objetivo final es la obtención de un esquema conceptual del genoma independiente de la especie. Esto permitiría generalizar todavía más el proceso de análisis genómicos. Este acercamiento presenta un reto muy interesante puesto que cada genoma contiene unos datos distintos. En parte debido a que el objetivo de secuenciación varía enormemente de una especie a otra.

Por último, se plantea la necesidad de incrementar la exigencia de la validación del trabajo. Esto se consigue mediante la adición de nuevos métodos. El primero de ellos es el experimento de caso único. Son experimentos cuyo objetivo es testear el prototipo. Para ello, el investigador ejecuta casos de prueba que representan posibles escenarios reales extraídos del contexto de uso. Es posible definir distintos casos de uso: ejemplo simple en el laboratorio, ejemplo real en el laboratorio o ejemplo real en el mundo real. Si las pruebas se realizan en el laboratorio es necesario que estas sean realizadas por personal ajeno al desarrollo del prototipo. Por cada caso deben describirse las tareas a realizar, identificar las métricas de interés y cómo se cuantificarán y determinar si se proporcionará un cuestionario posterior al experimento o no. El segundo de ellos es la investigación de acción técnica (TAR por sus siglas e inglés). Similar al experimento de caso único, su diferencia radica en el contexto puesto que se trata de un proyecto de consultoría en el mundo real. Se proporciona el artefacto a los usuarios en un contexto real, en condiciones de no controladas y se obtiene conocimiento a partir de la experiencia.

Bibliografía

[1] G. A. Wu, J. Terol, V. Ibanez, A. López-García, E. Pérez-Román, C. Borredá, C. Domingo, F. R. Tadeo, J. Carbonell-Caballero, R. Alonso *et al.*, "Genomics of the origin and evolution of citrus," *Nature*, vol. 554, no. 7692, p. 311, 2018.

[2] R. J. Wieringa, *Design science methodology: For information systems and software engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. [Online]. Available: http://link.springer.com/10.1007/978-3-662-43839-8

[3] W. Bateson and G. Mendel, *Mendel's principles of heredity*. Courier Corporation, 2013.

[4] "EDAM bioinformatics operations, types of data, data formats, identifiers, and topics - Scaffolding - Classes | NCBO BioPortal." [Online]. Available: http://bioportal.bioontology.org/ontologies/EDAM?p=classes{&}conceptid=operation{_}3216

[5] "The cost of sequencing a human genome." [Online]. Available: https://www.genome.gov/sequencingcosts/

[6] "Genetics and Genomics Science." [Online]. Available: https://www.genome.gov/19016904/

[7] "National Human Genome Research Institute (NHGRI) - Genome." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=90

[8] "National Human Genome Research Institute (NHGRI) - Chromosome." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=33

[9] "National Human Genome Research Institute (NHGRI) - Gene." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=70

[10] "National Human Genome Research Institute (NHGRI) - Intron." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=113

[11] "National Human Genome Research Institute (NHGRI) - Exon." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=61

[12] "National Human Genome Research Institute (NHGRI) - DNA." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=48

[13] "National Human Genome Research Institute (NHGRI) - RNA." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=180

[14] "National Human Genome Research Institute (NHGRI) - mRNA." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=123

[15] "National Human Genome Research Institute (NHGRI) - Ribosome." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=178

[16] "National Human Genome Research Institute (NHGRI) - Amino Acid." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=5

[17] "National Human Genome Research Institute (NHGRI) - Protein." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=169

[18] "National Human Genome Research Institute (NHGRI) - Biological Pathways Fact Sheet." [Online]. Available: https://www.genome.gov/27530687/biological-pathways-fact-sheet/

[19] "National Human Genome Research Institute (NHGRI) - Mutación." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=134

[20] "National Human Genome Research Institute (NHGRI) - Polimorfismo." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=160

[21] "National Human Genome Research Institute (NHGRI) - SNP." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=185

[22] "National Human Genome Research Institute (NHGRI) - Mutación sin sentido." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=138

[23] "National Human Genome Research Institute (NHGRI) - Mutación puntual." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=156

[24] "National Human Genome Research Institute (NHGRI) - Mutación con cambio de sentido." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=127

[25] "National Human Genome Research Institute (NHGRI) - Mutación con cambio en marco de lectura." [Online]. Available: https://www.genome.gov/glossary/index.cfm?id=68

[26] O. Pastor, S. España, J. I. Panach, and N. Aquino, "Model-driven development," *Informatik-Spektrum*, vol. 31, no. 5, pp. 394–407, Oct 2008. [Online]. Available: https://doi.org/10.1007/s00287-008-0275-8

[27] "BLAST TOPICS." [Online]. Available: https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web{&}PAGE{_}TYPE=BlastDocs{&}DOC{_}TYPE=BlastHelp

[28] "Specifications/gff3.md · The-Sequence-Ontology/Specifications." [Online]. Available: https://github.com/The-Sequence-Ontology/Specifications/blob/master/gff3.md

[29] Samtools, "The Variant Call Format Specification v4.3," *Online Resource*, pp. 1–28, 2015. [Online]. Available: https://samtools.github.io/hts-specs/VCFv4.3.pdf

[30] T. Smith and M. Waterman, "ªidentification of common molecular subsequences. º j," *Molecular Biology*, vol. 147, pp. 195–197, 1981.

[31] G. A. Wu, S. Prochnik, J. Jenkins, J. Salse, U. Hellsten, F. Murat, X. Perrier, M. Ruiz, S. Scalabrin, J. Terol *et al.*, "Sequencing of diverse mandarin, pummelo and orange genomes reveals complex history of admixture during citrus domestication," *Nature biotechnology*, vol. 32, no. 7, p. 656, 2014.

[32] "GATK | Home." [Online]. Available: https://software.broadinstitute.org/gatk/

[33] R. A. Fisher, "On the interpretation of χ^2 from contingency tables, and the calculation of p," *Journal of the Royal Statistical Society*, vol. 85, no. 1, pp. 87–94, 1922. [Online]. Available: http://www.jstor.org/stable/2340521

[34] P. Cingolani, A. Platts, M. Coon, T. Nguyen, L. Wang, S. Land, X. Lu, and D. Ruden, "A program for annotating and predicting the effects of single nucleotide polymorphisms, snpeff: Snps in the genome of drosophila melanogaster strain w1118; iso-2; iso-3," *Fly*, vol. 6, no. 2, pp. 80–92, 2012.

[35] "Codd's Twelve Rules | Department of Electronics, Computing & Mathematics." [Online]. Available: https://computing.derby.ac.uk/c/codds-twelve-rules/

[36] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, 6th ed. USA: Addison-Wesley Publishing Company, 2010.

[37] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in *Proceedings - 2011 6th International Conference on Pervasive Computing and Applications, ICPCA 2011*, jun 2011, pp. 363–366. [Online]. Available: http://arxiv.org/abs/1307.0191

[38] "AWS | Servicio de base de datos gestionada NoSQL (DynamoDB)." [Online]. Available: https://aws.amazon.com/es/dynamodb/

[39] "Apache Cassandra." [Online]. Available: http://cassandra.apache.org/

[40] "Reinventando la gestión de datos | MongoDB." [Online]. Available: https://www.mongodb.com/es

[41] "ObjectDB - Fast Object Database for Java with JPA/JDO support." [Online]. Available: http://www.objectdb.com/

[42] "The Neo4j Graph Platform – The #1 Platform for Connected Data." [Online]. Available: https://neo4j.com/

[43] N. F. McPhee, D. Donatucci, and T. Helmuth, "Using graph databases to explore the dynamics of genetic programming runs," in *Genetic Programming Theory and Practice XIII*. Springer, 2016, pp. 185–201.

[44] I. Balaur, A. Mazein, M. Saqi, A. Lysenko, C. J. Rawlings, and C. Auffray, "Recon2Neo4j: applying graph database technologies for managing comprehensive genome-scale networks," *Bioinformatics*, vol. 33, no. 7, pp. 1096–1098, 2016.

[45] P. Pareja-Tobes, E. Pareja-Tobes, M. Manrique, E. Pareja, and R. Tobes, "Bio4J: An Open source biological data integration platform." in *IWBBIO*, 2013, p. 281.

[46] I. Balaur, M. Saqi, A. Barat, A. Lysenko, A. Mazein, C. J. Rawlings, H. J. Ruskin, and C. Auffray, "EpiGeNet: A Graph Database of Interdependencies Between Genetic and Epigenetic Events in Colorectal Cancer," *Journal of Computational Biology*, vol. 24, no. 10, pp. 969–980, oct 2017. [Online]. Available: http://online.liebertpub.com/doi/10.1089/cmb.2016.0095

[47] D. Donatucci and M. K. Dramdahl, "Analysis of Ancestry in Genetic Programming with a Graph Database," no. April, pp. 1–21, 2014. [Online]. Available: https://pdfs.semanticscholar.org/2f74/ceea965f2aa11c9a693b7085f67123f42729.pdf

[48] R. J. Wilson, *Introduction to graph theory*, 5th ed. Prentice Hall, 2015.

[49] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*, 2nd ed. O'Reilly Media, Inc., 2015.

[50] R. Wieringa, "Empirical research methods for technology validation: Scaling up to practice," *Journal of systems and software*, vol. 95, pp. 19–31, 2014.

[51] J. F. Reyes Román, "Diseño y desarrollo de un sistema de información genómica basado en un modelo conceptual holístico del genoma humano," Ph.D. dissertation, 2018.

[52] S. A. Conger, *The new software engineering*. Wadsworth Pub, 1994.

[53] A. Pouloudi, "Stakeholder analysis as a front-end to knowledge elicitation," *AI & Society*, vol. 11, no. 1-2, pp. 122–137, mar 1997. [Online]. Available: http://link.springer.com/10.1007/BF02812443

Acrónimos

Símbolos

5′ cap

región no traducida cinco prima. 11, 29, 48, 75

A

A

Adenina. 12, 44, 127

ACID

Atomicidad, Consistencia, Integridad y Durabilidad. 53–57

ADN

Ácido Desoxirribonucleico. 3, 10–15, 24, 31, 36, 38, 123, 124, 126, 128

ADNc

ADN complementario. 44

ANN

ANN. 42, 44, 45, 71

API

Interfaz de programación de aplicaciones, del inglés *Application Programming Interface*. 6, 23, 80, 90, 93–95, 100–102

ARN

Ácido Ribonucleico. 12, 13, 123, 126

B

BASE

Disponibilidad Básica, *Soft-state*, consistencia eventual. 57

C

C

Citosina. 12, 15

CDS

región codificante. 12, 28–30, 41, 48, 50, 66, 73, 75, 81, 102, 106, 107

CSV

Fichero de valores separados por coma, del inglés *comma-separated values*. 62, 63, 65, 67, 77, 78, 100

D

DOM

Modelo de Objetos del Documento. 96

E

ECGCit.

Esquema Conceptual del Genoma de los Cítricos. 5, 19, 23, 46, 47, 62, 111–113

ETL

Extracción, Transformación y Carga. 6, 23, 46, 52, 62, 63, 111

G

G

Guanina. 12

GFF3

Genetic Feature Format Version 3. 24, 26

I

IVIA

Instituto Valenciano de Investigaciones Agrarias. 4, 5, 16–19, 23, 36, 40, 51, 106–112

J

JAR

Java ARchive. 41

JSON

JavaScript Object Notation. 55, 91

JWT

JSON Web Tokens. 95

L

LOF

LOF. 42, 45, 46, 71

M

MDD

desarrollo dirigido por modelos. 18

mRNA

ARN mensajero. 12, 13, 25, 26, 28–31, 33–35, 42, 44–46, 48, 50, 66, 68, 69, 71–76, 107, 108, 127, 129

N

NMD

NMD. 42, 45, 46, 71

O

OLTP

Procesamiento de Transacciones en Línea. 54

OOP

Programación orientada a objetos. 55

P

PHG

Proyecto del Genoma Humano. 10

PIM

Modelo Independiente de la Plataforma. 5

Poly-A

Poliadenilación. 12

PROS

Centro de Investigación en Métodos de Producción de Software. 17

PSM

Modelo Específico de la Pataforma. 5

S

SGBD

Sistema de Gestión de Bases de Datos. 6, 23, 52, 54–58

SGBDR

Sistema de Gestión de Bases de Datos Relacionales. 52, 53, 55

SNP

Polimorfismo de Nucleótido Sencillo. 36, 125

SQL

Lenguaje de Consulta Estructurada. 53

STR

Short Tandem Repeat. 37

T

T

Timina. 12, 15

TFM

Tésis de Fin de Máster. 3, 5, 7, 12, 17, 18, 112

U

U

Uracilo. 12

UTR 3'

región no traducida tres prima. 12, 28–30, 48, 66, 73, 75

UTR 5'

caperuza cinco prima. 12, 28, 30, 66, 73

V

VCF

Variant Call Format. 24, 36, 38, 40–43, 45, 46, 63, 64, 69, 71, 75, 76, 107, 131, 132

X

XML

eXtensible Markup Language. 55

Glosario

A

adyacencia libre de índices

Aquellos nodos que estén conectados entre sí deben enlazarse físicamente. 58, 59

aminoácido

Moleculas que forman parte de las proteínas. 11, 13, 15, 41, 66, 123–126, 128

ANN

Anotaciones funcionales de la variación. 119

B

bases

Compuestos orgánicos cíclicos que incluyen dos o más átomos de nitrógeno. Son parte fundamental de los nucleósidos, nucleótidos, nucleótidos cíclicos, dinucleótidos y ácidos nucleicos. . 11, 12, 14, 15, 84, 85

C

caperuza cinco prima

Es la región que se encuentra antes de la región codificante. En ciertas ocasiones, esta región puede ser traducida parcial o totalmente junto con la región codificante. 12, 122

centrómero

Región encogida del cromosoma que lo divide en un brazo largo (dos tercios de la longitud total aproximadamente) y un brazo corto (un tercio). 11

codón

Secuencia de tres nucleótidos de ADN o ARN que corresponde a un aminoácido específico. 12, 13, 15, 27, 42, 44, 126, 128

corte y empalme

Proceso post-transcripcional en la maduración del ARN donde se eliminan fragmentos de secuencias. En ocasiones se pueden obtener distintas isoformas de ARN debido a un proceso llamado empalme alternativo.. 107

cromosoma

Conjunto ordenado de ADN presente en el núcleo de la célula. 11, 12, 24, 76, 107

cromátida

Cada una de las dos hebras de un cromosoma ya duplicado. 11

D

diploide

Organismo que presenta dos juegos de cromosomas homólogos en sus células. 26, 37

dominio

Secuencia de aminoácidos en una proteína que está asociada con una función particular o segmento correspondiente de ADN. 31, 32, 51, 65, 72, 83, 86, 92, 93, 101, 108, 109

E

enzima

Tipo de proteína encargada de catalizar reacciones químicas. 13, 14, 17, 33–35, 48, 51, 66, 67, 73, 74, 83, 86, 92, 93, 101, 106–109

exón

Porción de un gen que codifica aminoácidos. 11–13, 26, 28–30, 42, 43, 48, 50, 66, 73, 75, 81, 102

F

framework

Marco de trabajo o conjunto estandarizado de conpceptos, prácticas y criterios dirigidos a enfocar un tipo de problema particular. 8

G

gen

Unidad física básica de la herencia. 4, 11–13, 17, 26, 28–30, 32, 33, 35, 42, 44–46, 48, 50, 51, 66, 68, 69, 71–76, 81–83, 85, 87, 92–95, 106, 107, 109

genoma

Conjunto de instrucciones genéticas presentes en una célula. 10, 11, 14, 17, 26, 51

genotipo

Conjunto de genes que existen en el núcleo celular. 16, 17, 38, 51

genética

Se refiere a un término que define el estudio de los genes y su rol en la herencia: cómo ciertos atributos o condiciones son heredados de una generación a otra. La genética engloba el estudio científico de los genes y sus efectos. Estos genes contienen las instrucciones para fabricar proteínas las cuales afectan de manera directa a las células y las funciones que estas desempeñan en nuestro cuerpo. 3, 11

genómica

Es un término más reciente y define el estudio del genoma completo, incluyendo las interacciones de los genes entre sí y con su entorno. Un ejemplo sería el estudio de enfermedades complejas como enfermedades cardíacas, diabetes o asma; ya que estas enfermedades están causadas por la interacción de diversos genes entre ellos y con factores ambientales. 3, 4, 11

genómico-cítrico

De la genómica, específicamente en el domínimo de los cítricos. 4, 5, 18, 19, 111

grafo

Conjunto de vértices (nodos) unidos por enlaces (aristas) que representan relaciones binarias entre un conjunto de elementos. 56, 57

grupo ortólogo

Grupos de genes pertenecientes a distintas especies que han evolucionado a partir de un ancestro común. Normalmente, estos genes mantienen la misma función a través del tiempo. 17, 32, 33, 67, 68, 74, 92, 93

H

haploide

Organismo que presenta un único juego de cromosomas homólogos en sus células. 26

haplotipo

Combinación de alelos que son transmitidos juntos. También identifica un conjunto de SNP que están estadísticamente asociados. 37

I

indel

Tipo de mutación que es mezcla de una inserción y una deleción. Se trata de una diferencia de longitud entre dos alelos donde no se puede saber si se produjo por una inserción o una deleción. 36, 42

intrón

Porción de un gen que no codifica aminoácidos. 11, 43, 48, 50, 81, 82, 102

L

LOF

Anotaciones que predicen pérdida de funcionalidad causada por la variación. 121

M

micro ARN

ARN monocatenario de una longitus de entre ceinte y veinticinco nucleótidos con capacidad de regulación génica. 43

motif

Patrón de secuencia de ADN altamente extendido y/o con posibilidad de tener significancia biológica. 43

mutación

Una mutación es un cambio en la secuencia del ADN. Las mutaciones pueden ser el resultado de errores en la copia del ADN durante la división celular, la exposición a radiaciones ionizantes o a sustancias químicas denominadas mutágenos, o infección por virus. Las mutaciones de la línea germinal se producen en los óvulos y el esperma y puede transmitirse a la descendencia, mientras que las mutaciones somáticas se producen en las células del cuerpo y no se pasan a los hijos. 14, 15, 126

Mutación con cambio de sentido

Se produce cuando el cambio de un solo par de bases da lugar a la sustitución de un aminoácido en la proteína resultante. 15

Mutación con cambio del marco de lectura

Inserción o deleción de un nucleótido en el que el número de pares de bases eliminado no es divisible por tres. 15

Mutación puntual

Tipo de mutación que afecta a tan sólo un par de bases. 15

Mutación sin sentido

Sustitución de un solo par de bases que da lugar a la aparición de un codón de terminación donde previamente había un codón que codificaba para un aminoácido. 15

mutágeno

Un mutágeno es un agente químico o físico, como las radiaciones ionizantes, que promueve los errores en la replicación del ADN. La exposición a un mutágeno puede producir mutaciones en el ADN que causan o contribuyen a enfermedades como el cáncer. 14

N

NMD

Anotaciones que predicen desintegramiento mediado sin sentido. Este tipo de anotación refiere a la degradación de mRNA que incluyen codones de terminación prematuros para evitar la creación de proteínas defectuosas. 121

NoSQL

Not Only SQL. Diverso conjunto de bases de datos que no siguen el modelo relacional. 52, 54–57

nucleótido

Un nucleótido es la pieza básica de los ácidos nucleicos. El ARN y el ADN son polímeros formados por largas cadenas de nucleótidos. Un nucleótido está formado por una molécula de azúcar (ribosa en el ARN o desoxirribosa en el ADN) unido a un grupo fosfato y una base nitrogenada. Las bases utilizadas en el ADN son la adenina (A), citosina (C), guanina (G) y timina (T). En el ARN, la base uracilo (U) ocupa el lugar de la timina. 11, 12, 15, 24, 26, 48, 126

O

objetivo de conocimiento

Un objetivo de conocimiento se encarga de describir y explicar fenómenos. 18

objetivo de diseño de artefacto

Un objetivo de diseño de artefacto, u objetivo de investigación técnica, se encarga de eliminar o reducir limitaciones para mejorar problemas existentes en el contexto social. 18

P

pathway

Conjunto de acciones producidas en la célula que produce un nuevo producto o realiza un cambio en esta. 14, 17, 33–35, 48, 68, 74, 75, 83, 86, 87, 92, 93, 101, 106–108

Poliadenilación

Adición de múltiples bases de A. Es importante para para la exportación nuclear y mejora la estabilidad del ARN mensajero. Esta sección se degrada con el tiempo hasta que el ARN mensajero se degrada enzimáticamente. 12, 121

polimorfismo

Implica una o más variantes de una secuencia particular de ADN. El tipo más común de polimorfismo implica la variación en un solo par de bases. Los polimorfismos también pueden ser de mucho mayor tamaño implicando largos tramos de ADN. 15, 128

Polimorfismo de Nucleótido Sencillo

Los polimorfismos de nucleótido único (SNP) son un tipo de polimorfismo que producen una variación en un solo par de bases. 15, 36, 42, 122

proteína

Molécula formada por un conjunto de aminoácidos dispuestos de una manera concreta. Existen miles de proteínas con funciones muy distintas. 3, 4, 12–15, 17, 24–26, 30–32, 41, 42, 48, 50, 51, 69, 83, 92, 93, 106–109, 123, 126, 128

R

rare amino acid

Se trata de aminoácidos que no se incorporan a proteínas. 41

región codificante

Es la región que codifica una proteína. 12, 27, 41, 120

región no traducida cinco prima

Nucleótido alterado situado al final de la región cinco prima. Tiene vital importancia en el proceso de creación de ARN mensajero estable y maduro. 11, 119

región no traducida tres prima

Es la región que se encuentra después de la región codificante, inmediatamente después del codón de terminación. Es frecuente encontrar regiones regulatorias que influencia la expresión génica en esta región. 12, 122

replicación

Proceso por el cual una molecula de ADN se duplica. 11, 128

ribosoma

Unidad celular encargada de la síntesis de proteínas. 12, 13

S

scaffold

Conjunto de secuencias no contiguas unidas cuya separación es de longitud conocida. 10, 24–27, 29, 30, 38, 40, 47, 51, 64, 66, 75, 76, 83, 87, 88, 94, 101, 102, 110, 141

segregación

En células eucariotas, proceso por el cual dos cromátidas formadas mediante replicación se unen a sus cromosomas homólogos. 11

Short Tandem Repeat

Secuencias de ADN en las que un fragmento se repite de manera consecutiva. 37, 122

sponsor

Encargados de otorgar el presupuesto para el proyecto. 8

stakeholder

A nivel general, un *stakeholder* es toda persona, organización o empresa que puede influenciar de manera directa o indirecta en una empresa o proyecto. Más específicamente, son todas aquellas personas y organizaciones afectadas por una aplicación[52][53]. 7–9, 17, 18, 49, 105, 113

T

teoría de grafos

Conjunto de vértices (nodos) unidos por enlaces (aristas) que representan relaciones binarias entre un conjunto de elementos. 56

tránscrito

MRNA obtenido después del proceso de transcripción previo al proceso de maduración. 43, 45

APÉNDICE A
Fragmentos de código

A.1 Línea de comandos

Página 41:

```
1     java -Xmx${memoria} -jar snpEff.jar ${base_de_datos} ${ruta_al_fichero_de_entrada} >
        ↪  ${ruta_al_fichero_de_salida}
```

Fragmento de Código A.1: Formato de las instrucciones del script para ejecutar herramienta SnpEff

Página 42:

```
1  #!/usr/bin/env bash
2
3   java -Xmx4g -jar snpEff.jar Cclementina ~/citricos/snpEff/vcf/ivia_000_indel.vcf >
      ↪  snpeff_000_indel.vcf;
4   java -Xmx4g -jar snpEff.jar Cclementina ~/citricos/snpEff/vcf/ivia_000_snp.vcf >
      ↪  snpeff_000_snp.vcf;
```

Fragmento de Código A.2: Extracto de script para ejecución de SnpEff

Página 64:

```
1  find . -name "*.*" -exec bgzip {} \
```

Fragmento de Código A.3: Comando para la compresión de ficheros VCF

Página 64:

```
find . -name "*.vcf.gz" -exec tabix -p vcf {} \
```

Fragmento de Código A.4: Comando para la el indexado de ficheros VCF

Página 78:

```
#!/usr/bin/env bash

NEO4J_DEBUG=true

sudo /usr/bin/neo4j-admin import \
--database=citricos.db \
--delimiter=";" \
--array-delimiter="," \
--ignore-duplicate-nodes=true \
--id-type=string \
--nodes "variation_headers.csv,variations.csv" \
--nodes "lecture_headers.csv,lectures.csv" \
--relationships "variation_lecture_headers.csv,variations_lectures.csv" \
```

Fragmento de Código A.5: Script create_db.sh

Página 77:

```
#!/usr/bin/env bash
time python3 ../nodes/domain.py;
time python3 ../nodes/elements.py;
time python3 ../nodes/enzyme.py;
time python3 ../nodes/orthologs.py;
time python3 ../nodes/pathway.py;
time python3 ../nodes_n_edges/go.py;
time python3 ../nodes_n_edges/proteins.py;
time python3 ../nodes_n_edges/variations.py;
time python3 ../edges/domain_gene.py;
time python3 ../edges/elements.py;
time python3 ../edges/enzyme_relations.py;
time python3 ../edges/ortholog_relations.py;
time python3 ../edges/pathway_enzyme.py;
time python3 ../edges/variation_elements.py;
time awk "!x[$0]++" ../files/out/csv/variation_elements.csv >
  ↪   ../files/out/csv/variation_elements_f.csv
time awk "!x[$0]++" ../files/out/csv/variations.csv > ../files/out/csv/variations_f.csv
time awk "!x[$0]++" ../files/out/csv/annotations.csv > ../files/out/csv/annotations_f.csv
time awk "!x[$0]++" ../files/out/csv/ann_var.csv > ../files/out/csv/ann_var_f.csv
time awk "!x[$0]++" ../files/out/csv/ann_gen.csv > ../files/out/csv/ann_gen_f.csv
time awk "!x[$0]++" ../files/out/csv/ann_mrna.csv > ../files/out/csv/ann_mrna_f.csv
time awk "!x[$0]++" ../files/out/csv/lof.csv > ../files/out/csv/lof_f.csv
time awk "!x[$0]++" ../files/out/csv/lof_var.csv > ../files/out/csv/lof_var_f.csv
time awk "!x[$0]++" ../files/out/csv/lof_gen.csv > ../files/out/csv/lof_gen_f.csv
time awk "!x[$0]++" ../files/out/csv/nmd.csv > ../files/out/csv/nmd_f.csv
time awk "!x[$0]++" ../files/out/csv/nmd_var.csv > ../files/out/csv/nmd_var_f.csv
time awk "!x[$0]++" ../files/out/csv/nmd_gen.csv > ../files/out/csv/nmd_gen_f.csv
```

Fragmento de Código A.6: Script create_model.sh

Página 78:

```
IMPORT DONE in 22m 53s 931ms.
Imported:
  278082816 nodes
  465146788 relationships
  1743968582 properties
Peak memory usage: 5.64 GB
```

Fragmento de Código A.7: Resultado creación base de datos

Página 80:

```
$ echo 'deadline' > /sys/block/sda/queue/scheduler
$ cat               /sys/block/sda/queue/scheduler
noop [deadline] cfq
```

Fragmento de Código A.8: Cambio de planificador del sistema operativo

A.2 Configuración

Página 41:

```
1       # Databases & Genomes
2       #
3       # One entry per genome version.
4       #
5       # For genome version 'ZZZ' the entries look like
6       #   ZZZ.genome : Real name for ZZZ (e.g. 'Human')
7       #
8       # Clementona
9       Cclementina.genome: clementona
```

Fragmento de Código A.9: Fichero snpEff.config

A.3 Cypher

Página 94:

```
WITH ["901","902"] AS B
MATCH (l:`10-80`)-[:lecture_from]->(v:variation)<-[:ann]-(a:annotation)-[:annotate_gene]
  ↪   ->(g:gene)
WHERE l.Variety IN B
AND a.annotation_impact IN ["HIGH"]
AND (l.AB >= 0.8)
AND EXISTS ((v)-[:variarion_in_cds]->(:cds))
WITH v, count(distinct l) AS cnt
WHERE cnt > 0
WITH ["000","003"] AS A, COLLECT(v) AS notVarieties
MATCH (l:`10-80`)-[:lecture_from]->(v:variation)<-[:ann]-(a:annotation)-[:annotate_gene]
  ↪   ->(g:gene)
USING INDEX l:`10-80`(Variety)
WHERE l.Variety IN A
AND a.annotation_impact IN ["HIGH"]
AND (l.AB >= 0.8)
AND EXISTS ((v)-[:variarion_in_cds]->(:cds))
AND NOT v IN notVarieties
WITH v, count(distinct l) AS cnt, size(A) AS incnt, g, COLLECT(distinct a.annotation) AS a,
  ↪   COLLECT(DISTINCT {key: l.Variety, value: l.AB}) AS l
WHERE cnt >= 2
RETURN v.id AS v, a Sa, g.id AS g, l AS vs
```

Fragmento de Código A.10: Consulta generada para /q1

Página 94:

```
MATCH (g:gene {id: "Ciclev1000235m.g.v1.0"}) RETURN g.scaffold * 10^9 + g.start - 5000 AS
  ↪   start, g.scaffold * 10^9 + g.end + 5000 AS end
```

Fragmento de Código A.11: Generación automática de consultas para obtención de posiciones de genes

Página 94:

```
MATCH (v:variation)
WHERE 1023522053 <= v.pos_abs <= 1023547559
WITH COLLECT(v) AS v1
UNWIND v1 AS v
WITH v, ["901","902"] AS B, v AS vi
MATCH (l:`10-80`)-[:lecture_from]->(v)<-[:ann]-(a:annotation)
WHERE l.Variety IN B
AND a.annotation_impact IN ["HIGH","LOW","MODERATE","MODIFIER"]
AND (l.AB > 0.2)
WITH v, count(DISTINCT l) AS cnt
WHERE cnt > 0
WITH
CASE
WHEN size(COLLECT(v)) = 0 THEN []
else COLLECT(v)
END  AS notVarieties
WITH notVarieties, ["000","003"] AS A
MATCH (v:variation)
WHERE 1023522053 <= v.pos_abs <= 1023547559
AND NOT v IN notVarieties
WITH notVarieties, COLLECT(v) AS v1, A
UNWIND v1 AS v
MATCH (l:`10-80`)-[:lecture_from]->(v)<-[:ann]-(a:annotation)-[:annotate_gene]->(g:gene)
WHERE l.Variety IN A
AND a.annotation_impact IN ["HIGH","LOW","MODERATE","MODIFIER"]
AND (l.AB > 0.2)
WITH v, count(DISTINCT l) AS cnt, size(A) AS incnt, g, COLLECT(DISTINCT a.annotation) AS a,
  ↪   COLLECT(DISTINCT {key: l.Variety, value: l.AB}) AS l
WHERE cnt >= 2
RETURN v.id AS v, a AS a, g.id AS g, l AS vs
```

Fragmento de Código A.12: Consulta generada para /by_pos

Página 94

```
MATCH (l:`10-90`)-[:lecture_from]->(v:variation)<-[:ann]-(a:annotation)-[:annotate_gene]
  ↪   ->(g:gene)
WHERE 1026000000 <= v.pos_abs <= 1026000500
AND (l.AB > 0.2)
AND a.annotation_impact IN ["HIGH","LOW","MODERATE","MODIFIER"]
RETURN v.id AS v, g.id AS g, COLLECT(DISTINCT a.annotation) AS a, COLLECT(DISTINCT {key:
  ↪   l.Variety, value: l.AB}) AS vs
```

Fragmento de Código A.13: Consulta generada para /by_range

A.4 JavaScript

Página 91:

```
global.dictionary = await require('./../lib/neo4j/dataDictionary').dataTable();
```

Fragmento de Código A.14: Obtención del diccionario de datos

Página 92:

```
exports.dataTable = function () {
    const promises = [genes.geneGO(), genes.geneDomain(), genes.geneProtein(),
      ↪  genes.geneOrtholog(), genes.geneEnzyme(), enzymes.enzymePathway()];
    const names = ['geneGO', 'geneDomain', 'geneProtein', 'geneOrtholog', 'geneEnzyme',
      ↪  'enzymePathway'];

    return new Promise((resolve, reject) => {
        Promise.all(promises)
            .then(results => {
                resolve(zipObject(names, results));
            })
            .catch(error => reject(error))
    });
};
```

Fragmento de Código A.15: Extracto de fichero dataDictionary.js

Página 92:

```
exports.xCollectsY = function (x, y, z, id = 'id') {
    ...
    const query = `match (x:${x})-[:${y}]-(y:${z}) return x.${id} as ${x}, collect(distinct y)
      ↪   as ${z}s`;
    const result = {};
    ...
    return new Promise((resolve, reject) => {
        session.run(query).subscribe({
            onNext: record => {
                result[record.get(x)] = record.get(`${z}s`).map(e => e.properties)
            },
            onCompleted: () => {
                session.close(); driver.close(); resolve(result);
            },
            onError: (error) => {
                session.close(); driver.close(); reject(error);
            }
        });
    });
};
```

Fragmento de Código A.16: Función generadora de generators.js

Página 93:

```
exports.QueryGenerator = function({A, B, filter = 'lecture', annotation_impact = ['HIGH',
  ↪   'LOW', 'MODIFIER', 'MODERATE'], min = A.length, max = 0, depth = 0, AB}) {
    ...
    function setGroup(query, group, filter, letter) {...}
    function setAnnotationImpact(query, annotation_impact) {...}
    function setAB(query, conditions) {...}
    function setDepth(query, depth) {...}
    function setFlexibility(query, number, type) {...}
    function setSetOperation(query) {...}
    function setReturn(query) {...}
    ...
};
```

Fragmento de Código A.17: Generación automática de consultas para /q1

Página 94:

```
exports.get_variations_by_position = function({A, B, filter = 'lecture', annotation_impact =
  ↪   ['HIGH', 'LOW', 'MODIFIER', 'MODERATE'], min = A.length, max = 0, AB, positions}) {
    ...
    function getVariations(query, positions, letter) {...}
    function setGroup(query, group, filter, letter) {...}
    function setAnnotationImpact(query, annotation_impact) {...}
    function setAB(query, conditions) {...}
    function setFlexibility(query, number, type) {...}
    function setReturn(query) {...}
    ...
};
```

Fragmento de Código A.18: Generación automática de consultas para /by_pos

Página 94:

```
exports.get_variations_by_chr_region = function({scaffold, start_pos, end_pos, filter, AB,
  ↪   annotation_impact = ['HIGH', 'LOW', 'MODIFIER', 'MODERATE']}) {
    ...
    function setMatch(query, filter) {...}
    function setRegion(query, scaffold, start_pos, end_pos) {...}
    function setAB(query, conditions) {...}
    function setAnnotationImpact(query, annotation_impact) {...}
    function setReturn(query) {...}
    ...
};
```

Fragmento de Código A.19: Generación automática de consultas para /reg_chr

Página 94

```
exports.runOne = (query, function_to_execute) => {
    ...
    return new Promise((resolve, reject) => {
        const start = new Date();
        session.run(query).subscribe({
            onNext: record => {
                const item = function_to_execute ? function_to_execute(record) : record;
                result.push(item);
            },
            onCompleted: () => {
                session.close(); driver.close();
                return resolve({
                    query: query,
                    result: result,
                    count: [...new Set(result.map(item=>item.v))].length,
                    time: new Date() - start
                });
            },
            onError: error => {
                session.close(); driver.close(); return reject(error)
            }
        })
    });
};
```

Fragmento de Código A.20: Obtención del diccionario de datos

Página 95

```
exports.formatVariationDataTable = function(record) {
    const gene = record.get('g'); const enzyme = dictionary['geneEnzyme'][gene] || [];

    return {
        v: record.get('v'),
        ab: record.get('vs').map(e => e['value']),
        annotation: pipe(record.get('a'), flattenDeep, uniq),
        gene: gene,
        go: dictionary['geneGO'][gene] || [],
        domain: dictionary['geneDomain'][gene] || [],
        enzyme: enzyme,
        pathway: pipe(enzyme, enzymes => enzymes.map(enzyme =>
          ↪   dictionary['enzymePathway'][enzyme.id]), flattenDeep, uniq),
        varieties: record.get('vs').map(e => e['key'])
    };
};
```

Fragmento de Código A.21: Función para dar formato a los registros de variaciones

Página 102

```
public consulta1(path, body) {
    return this.tokenService.getToken()
      .then(token => {
        const fullUrl = `${environment.APIuri}${path}`;
        return this.http.post(fullUrl, body, { headers: {
            'content-type': 'application/json',
            'Authorization': `${token['token_type']} ${token['access_token']}`
          }}).toPromise();
      })
      .catch(error => {
        return Promise.reject(error);
      });
  }
```

Fragmento de Código A.22: Método consulta1 del servicio VarietiesService

Página 103

```
export interface Q1ResponseTable {
  v: string; ab: number[]; annotation: string[]; gene: string;
  go: { type: string; description: string; id: string; }[];
  domain: { description: string; id: string; }[];
  enzyme: { n_seq: { low: number; high: number }; description: string; id: string; }[];
  pathway: { description: string; id_map?: string; id_ko?: string; }[];
  varieties: string[];
}
```

Fragmento de Código A.23: Interfaz Q1ResponseTable

A.5 Python

Página 66:

```
def optimize_sequence(self) -> dict:
  result = dict()
  with gzip.open(self.sequencePath, 'rt') as file:
    identifier = None
    seq = []
    for line in file:
      if line.startswith(">scaffold_"):
        if identifier is not None:
          result[identifier] = ''.join(seq)
        identifier = line[1:]
        seq = []
      seq.append(str(line))
  return result
```

Fragmento de Código A.24: Optimización se secuencias de scaffolds en elements.py

Página 70:

```
shared = dict()
shared['queue'] = Queue()
shared['queueLock'] = Lock()
shared['v_count'] = Value('i',-1)
shared['vLock'] = Lock()
shared['ioLoop'] = Value('b', True)
shared['ioLock'] = Lock()

ioProcess = Process(target=self.io_loop, args=(shared,))
ioProcess.start()
for file in self.files:
  basename = file.split('/')[-1][:-3]
  self.load_and_write(file, basename, shared, )

with shared['ioLock']:
  shared['ioLoop'].value = False
try:
  ioProcess.join()
except:
  pass
```

Fragmento de Código A.25: Fragmento método main script del variations.py

Página 71:

```
def io_loop(self, shared) -> None:
  files = self.get_files()
  writers = self.get_writers(files)

  while True:
    with shared['ioLock']:
      if not shared['ioLoop'].value: break
    queueLength = shared['queue'].qsize()
    if queueLength != 0:
      shared['queueLock'].acquire()
      chunkwriters = list()
      for i in range(min(16,queueLength)):
        chunkwriters.append(shared['queue'].get())
      shared['queueLock'].release()
      self.write_chunks(chunkwriters,writers)
      del chunkwriters
    time.sleep((0.5,0)[queueLength>1])
  queue_size = shared['queue'].qsize()

  if queue_size != 0:
    shared['queueLock'].acquire()
    chunkwriters = list()
    for i in range(queue_size):
      chunkwriters.append(shared['queue'].get())
    shared['queueLock'].release()
    self.write_chunks(chunkwriters,writers)
    del chunkwriters

  for csvName in csvNames: files[csvName].close()
```

Fragmento de Código A.26: Fragmento método io_loop script del variations.py

Página 72:

```
def get_mrna_id(self) -> None:
  with gzip.open(self.mrnaIDsPath, 'rt') as file:
    for _ in range(2):
      next(file)
    for line in file:
      line = line.strip().split('\t')
      if line[2] == 'mRNA':
        line = dict(x.split('=') for x in line[8].split(';'))
        self.ids[line['Name']] = [line['ID'], line['Parent']]
```

Fragmento de Código A.27: Método *get_mrna_id*

Página 74:

```
with gzip.open(self.file, 'rt') as file:
  for line in file:
    encimas = []
    line = line.split("\t")
    if 'Ciclev' not in line[0] or 'HSA' in line[2]:
      continue
    if '[' in line[2] and 'EC' in line[2]:
      encimas = ['ec:' + x for x in line[2].rpartition('[')[-1].partition(']')[0][3:].split('
        ↪   ')]
    if len(encimas) > 0:
      for encima in encimas:
        writer_rel_enc.writerow([encima, line[1], 'enz_orth'])
    writer_rel_gene.writerow([self.ids[line[0]], line[1], 'gene_orth'])
```

Fragmento de Código A.28: Filtrado y escritura de datos en ortholog_relations.py

Página 76:

```
...
# Obtención de la lista de scaffolds
p = subprocess.Popen(["tabix","-l",file], stdout=subprocess.PIPE, universal_newlines=True)
...
# Obtención del scaffold
while True:
    chrom = p.stdout.readline().strip()
    if not chrom:
        break
    ...
    # Obtención de las variaciones de dicho scaffold
    records = vcf_reader.fetch(chrom)
...
```

Fragmento de Código A.29: Utilización de tabix en script variations.py