UNIVERSIDAD POLITECNICA DE VALENCIA

Tesis de Máster

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

Diagen: Modelado e Implementación de un Framework para el Análisis Personalizado del ADN

Febrero de 2011, Valencia

Maria José Villanueva Del Pozo

mvillanueva@pros.upv.es

Director: *Óscar Pastor López*

opastor@pros.upv.es

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Hacia la medicina personalizada

El análisis de secuencias de ADN ha permitido mejorar el conocimiento que tiene el ser humano sobre sí mismo, a fin de comprender por qué somos tan iguales y a la vez tan diferentes.

En el año 1990, el Human Genome Project [35] inició un proyecto para obtener la secuencia completa de nucleótidos del genoma humano. Desde ese momento, proliferó el interés por averiguar las relaciones que existían entre las moléculas que conforman el ADN de un individuo (genotipo) y sus características externas (fenotipo).

Los avances tecnológicos en secuenciación, como la reciente llamada "Next Generation Sequencing" [28], han permitido obtener más secuencias en menor tiempo y a menor precio. Estas nuevas tecnologías han permitido pasar de un coste de 60.000.000$/genoma, en 2006, a 1.000.000$/genoma, en 2007, hasta llegar a 640$/genoma en 2010. La previsión establece que en 2012 secuenciar 100.000 genomas costará 500$ y, en 2014, secuenciar 1.000.000 de genomas costará 100$ [3]. Estos avances han supuesto un gran incremento en el número de secuencias genómicas disponibles para el análisis y la investigación.

Gracias a esta explosión de datos genómicos, el conocimiento sobre las relaciones entre genotipo y fenotipo ha avanzado considerablemente. Por ejemplo, en el campo de la medicina, se planteó una linea de investigación para averiguar si el hecho de poseer una característica externa, concretamente una enfermedad, suponía tener también un genotipo específico. Estas investigaciones consistían en analizar el ADN de un conjunto de personas, y observar si los que padecían la misma enfermedad tenían una combinación genotípica específica que, sin embargo, no tenían los individuos sanos. Muchas de estas investigaciones revelaron que las hipótesis iniciales iban bien encaminadas y que sí existían combinaciones concretas en el genotipo asociadas a enfermedades.

Cuando se dice que una combinación genética concreta esta asociada, o provoca, una enfermedad, los biólogos se refieren a ella con el término mutación. Una mutación supone un

cambio en la secuencia de ADN que, como consecuencia, produce un cambio en la fabricación de proteínas. A causa de esta alteración química, se produce la enfermedad.

Posteriormente a estos descubrimientos, se ha establecido una nueva metodología para la diagnosis de enfermedades que consiste en inferir un fenotipo utilizando la relación genotipo-fenotipo de manera inversa. De esta forma, se buscan mutaciones en un individuo y, posteriormente, los fenotipos asociados. Por eso, cuando se conoce que ciertas combinaciones genotípicas provocan enfermedades [4, 8, 20] - tales como el cáncer, la diabetes o la neurofibromatosis- es posible alertar a un individuo que es propenso a padecerlas si se encuentran dichas combinaciones en su ADN.

Como consecuencia de estos hallazgos, en el campo de la farmacología, surgió la siguiente pregunta: si puede existir una relación entre el genotipo y una enfermedad, ¿es posible que una combinación genotípica concreta también afecte a la efectividad de un tratamiento o una medicación? De la misma manera que ocurrió con la diagnosis genética de enfermedades, las investigaciones revelaron que sí existía una relación entre el genotipo de un paciente y la efectividad de un tratamiento. Se había descubierto que algunos tratamientos podían ser beneficiosos, perjudiciales, o no tener ningún efecto, dependiendo del ADN de cada individuo [17].

Con estas dos premisas, se han abierto las puertas a la medicina genética personalizada, es decir, se plantea la posibilidad de utilizar, como complemento a la sintomología, la información genética de un paciente con la finalidad de realizar un diagnóstico más exacto. Una medicina donde la información genética de los individuos se utilice para diagnosticar enfermedades de origen genético, o para determinar, que medicamento, dosis y el momento de administración va a ser más efectivo para el paciente enfermo.

Como siguiente paso, se han abierto las puertas a la medicina genómica personalizada, basada en el análisis de todo el genoma. Una medicina donde ya no solo se realicen análisis genéticos para complementar diagnósticos de un paciente enfermo, sino que se analice el genoma completo de los individuos, sanos o enfermos, en busca de combinaciones genotípicas que puedan originar todo tipo de enfermedades de origen genético. Esta metodología permitiría reducir la necesidad de realizar algunas pruebas para el diagnóstico y aumentar la efectividad de los tratamientos. De ese modo, los tratamientos se aplicarían cuando la enfermedad aún no se hubiera expresado o en fases tempranas de la enfermedad, en lugar de aplicarlos, con menos posibilidades de cura, cuando la enfermedad ya estuviera avanzada. Las posibilidades de prevenir o curar enfermedades aumentarían y se podría ahorrar tiempo, dinero y conseguir mejoras en los tratamientos.

La medicina genética y la medicina genómica son dos formas de medicina que, aunque esperanzadoras, les falta mucho camino por recorrer para convertirse en una realidad. En la actualidad, se realizan análisis genéticos como un soporte opcional al diagnóstico, y se utiliza la farmacogenética para tratar un conjunto reducido de enfermedades. Los factores que influyen en la baja aplicación de esta metodología son:

1. El conocimiento del genoma humano aún es muy reducido: De momento no se conocen muchas de las asociaciones genotipo-fenotipo que aseguren la relación causa-efecto con

total seguridad. Por esta razón, el análisis genético como soporte al diagnóstico se utiliza para diagnosticar pocas enfermedades.

2. Los recursos de los que disponen los biólogos que se dedican a analizar e investigar las relaciones genotipo-fenotipo son limitados. Esta situación ralentiza considerablemente, tanto la ejecución de los análisis genéticos de soporte al diagnóstico, como los descubrimientos de nuevas relaciones genotipo-fenotipo.

Por estos motivos, el reto que se presenta en el campo de la bioinformática es impulsar el desarrollo de una infraestructura que de soporte a los análisis genéticos y genómicos para llegar a hacer de la medicina personalizada una realidad.

1.2. Motivación

El problema actual con el que se encuentran los investigadores de medicina genética, al llevar a cabo sus tareas, es la falta de herramientas y sistemas de información que les permitan ejecutarlas de manera eficiente.

Por este motivo, aquellos que disponen de conocimientos avanzados en informática, utilizan bases de datos o herramientas de análisis creadas por ellos mismos, la mayoría de ellas, implementados lejos de las buenas prácticas de la ingeniería de los sistemas de información y del desarrollo del software. Este soporte informático, aunque les sirve de ayuda, no es capaz de responder eficientemente ante tantos datos a procesar y a su gran complejidad.

Al igual que ocurre en otros dominios, se observa que existen diferentes niveles de conocimiento entre los biólogos y los informáticos. Los biólogos no conocen los principios sobre los que se deben construir los sistemas de información genómicos ni las herramientas software de análisis de secuencias. Del mismo modo, los informáticos no conocen las propiedades biológicas del ADN, ni los procesos biológicos que explican la traducción del ADN en proteínas, ni como estas proteínas pueden llegar a provocar una enfermedad. Por ese motivo, el desarrollo de sistemas informáticos que cubran las expectativas de los biólogos que las utilizan no es una tarea trivial.

El problema reside en lo que podríamos llamar la "nube del conocimiento biológico". Los biólogos que analizan las secuencias de ADN disponen de un conocimiento del que hacen uso diariamente pero que no saben explicar o transmitir a personas ajenas a su dominio. En otras ocasiones, simplemente no saben identificar la metodología que siguen para realizar ciertas tareas o llegar a ciertas conclusiones, es decir, saben que una cosa es así pero no saben explicar el porqué del procedimiento seguido.

Centrándonos en el diagnóstico genético basado en la búsqueda de mutaciones en secuencias genéticas de ADN para su caracterización fenotípica, este problema de falta de recursos también se manifiesta. Las herramientas y sistemas de información que se desarrollan, aunque son funcionales y les ayudan en sus tareas, no consiguen cumplir completamente sus expectativas

[25]. Esto a su vez deriva en la creación de más y más herramientas a fin de cubrir los requisitos que no se han cubierto inicialmente. Esta proliferación de herramientas no hace más que agravar su situación de descontento ya que, al final, los biólogos disponen de un gran abanico de posibilidades y no disponen de los criterios adecuados para discernir que herramienta es la más adecuada para cada tarea.

Ante esta situación, los investigadores intentan realizar el análisis genético con las limitadas herramientas de las que disponen, utilizando varias de ellas e incluso completando manualmente algunas tareas hasta obtener la ejecución del proceso completo. Para ello, deben realizar el flujo de datos entre varias herramientas, tarea que les consume mucho tiempo y que no siempre se trata de una tarea trivial debido a la falta de formalización ni estándares.

Con la finalidad de conseguir una mayor efectividad en la ejecución de las tareas de análisis genético y genómico es necesario el desarrollo de sistemas de información genómicos estructurados y herramientas software capaces de explotar dichos sistemas de información.

1.3. Contexto

Esta tesis de máster se ha desarrollado en el Centro de Investigación en Métodos de Producción de Software (PROS) de la Universidad Politécnica de Valencia.

Durante los últimos tres años, el Centro PROS ha enfocado algunas de sus lineas de investigación hacia el dominio de la bioinformática. En este punto, el proyecto de investigación planteado, llamado Diagen , se enfocará hacia la determinación de los mecanismos idóneos que permitan adaptar los procesos de diseño e implementación de software a las necesidades de cualquier proyecto basado en el dominio del genoma humano. El objetivo del proyecto será conseguir que estos diseños sean lo más robustos y fiables posible, a través de la aplicación de técnicas de Desarrollo del Software Dirigido por Modelos (DSDM) al ámbito del modelado del genoma humano.

La presente tesis de máster, englobado dentro del proyecto Diagen, aborda la problemática planteada por el Instituto de Medicina Genómica IMeGen (www.imegen.es), dedicada al análisis de variaciones genómicas en secuencias de ADN en humanos.

IMeGen es una empresa asociada a la Universidad de Valencia dedicada a la secuenciación y análisis de ADN como soporte al diagnóstico. A partir de la secuenciación de las muestras, los biólogos de IMeGen, buscan indicios de que el individuo secuenciado padezca una enfermedad de origen genético asociado a su genotipo, como puede ser el cáncer de mama u ovario, fibrosis quística, hemofilia, etc.

En sus trabajos diarios, los biólogos utilizan herramientas diseñadas para el análisis de secuencias genómicas: herramientas de limpieza de secuencias, herramientas de ensamblaje, herramientas de alineamiento, etc. El problema con el que se encuentran es que dichas herramientas no cubren sus necesidades y el análisis de solo un gen de un individuo se convierte una tarea difícil, manual y repetitiva, que incluso puede llegar a consumir horas o días de su trabajo.

Ante la necesidad de dar solución a estos problemas, surge el proyecto de investigación multidisciplinar (Diagen), donde colaboran investigadores informáticos y biólogos.

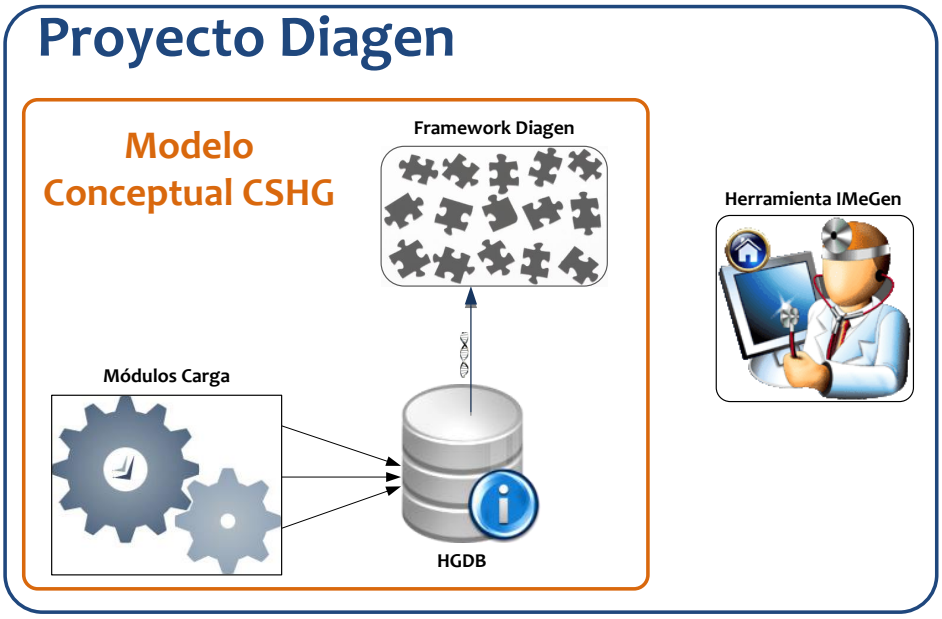


Figura 1.1: El proyecto Diagen

El primer paso que se debe afrontar dentro del marco del proyecto de investigación (Figura 1.1) es la formalización del dominio mediante la creación de un modelo conceptual que capture de forma concisa los conceptos relacionados con el genoma humano [23]. En esta fase del proyecto se ha definido y acotado los diferentes términos biológicos, tan ambiguos y heterogéneos hasta el momento. Como primera aproximación, se ha presentado una primera versión del modelo conceptual del genoma humano, denominado CSHG (Conceptual Schema of Human Genome). Esta primera versión se basa en los aspectos descriptivos del ADN, los procesos de transcripción de proteínas y la ocurrencia de variaciones genéticas en el ADN. Actualmente se está trabajando para ampliar esta versión para que incluya información adicional sobre los procesos biológicos que se producen en el ADN, y más detalles acerca de los efectos combinados de las variaciones genéticas.

El siguiente paso es el desarrollo de un sistema de información basado en dicho modelo conceptual. Con dicho fin, se ha creado una base de datos, HGDB (Human Genome Database), para que contenga, correctamente estructurada, la información genómica disponible en diferentes fuentes de datos dispersas y heterogéneas. Actualmente se está trabajando en módulos de carga para obtener la información genómica de diferentes fuentes de datos.

Como siguiente paso, el cual se aborda en la presente tesis de máster, se presenta el reto de utilizar dicho sistema de información basado en el modelo conceptual del genoma humano para la explotación de datos genómicos. Concretamente, se aborda el diseño de un framework de integración y explotación de datos genómicos para la detección de variaciones en secuencias genéticas de ADN y su caracterización fenotípica. La presente tesis de máster parte del desarrollo inicial de un prototipo, presentado en otra tesis de máster en el centro PROS, en la que se

desarrolla una primera aproximación al problema para identificar las dificultades del proceso y presentar las lecciones aprendidas que sirven como base para el desarrollo de la presente tesis de máster.

1.4. Objetivos

El objetivo general del proyecto Diagen es desarrollar de manera sistemática herramientas software para el análisis genético del ADN del ser humano. Para ello conseguir el objetivo general se establecen los objetivos parciales:

- Desarrollo de un sistema de información genómica
- Explotación del sistema de información para la extracción de conocimiento

La presente tesis de máster se centra en el segundo objetivo parcial y tiene como meta la detección y análisis de variaciones genéticas para su posterior caracterización fenotípica en enfermedades o en la ausencia de ellas.

El objetivo principal es la formalización del proceso mediante modelos conceptuales, con el fin de desarrollar una herramienta que sea capaz de cubrir completamente el proceso de análisis. El objetivo perseguido es reducir la barrera computacional con la que se encuentran los biólogos a la hora de realizar un análisis de una secuencias genética.

Como preguntas de investigación para conseguir dicho objetivo se plantean:

¿Cómo se puede formalizar el proceso de detección y análisis de variaciones genéticas de ADN para obtener su caracterización fenotípica?

Esta tesis de máster aborda la formalización del proceso biológico objetivo mediante modelos conceptuales. En primer lugar se debe realizar una especificación del problema con la colaboración de expertos en biología que trabajen en el análisis de variaciones genéticas. Para ello se presenta la especificación textual de requisitos del problema, el diagrama del proceso de negocio y las limitaciones encontradas en el proceso de análisis.

En segundo lugar, se debe establecer claramente los límites entre las diferentes tareas en las que se compone.

Por último se debe realizar un análisis de las herramientas biológicas disponibles para poder determinar las razones por la cuales no se están cubriendo las necesidades requeridas. Para ello, se ha estudiado las características de un conjunto de herramientas y determinado las ventajas y desventajas de cada una de ellas.

¿Cómo se deben desarrollar las herramientas de análisis de variaciones para que se ajusten totalmente a las necesidades de los biólogos que las utilizan?

A partir de la formalización del problema mediante modelos conceptuales, en esta tesis de máster se aborda el desarrollo de herramientas de análisis de manera sistemática.

En primer lugar se debe proporcionar un framework para la integración de herramientas biológicas que permita desarrollar herramientas a medida de las necesidades de los biólogos. Para ello, se desarrolla la implementación, basada en modelos conceptuales para soportar las fases identificadas.

En segundo lugar se debe definir una estrategia de implementación que sea fiel a los requisitos planteados para el desarrollo de la herramienta de análisis de variaciones genéticas. Para ello, presenta una estrategia de implementación basada en modelos conceptuales.

De esta manera la implementación final de la herramienta software reflejará todo el conocimiento adquirido y representado en los modelos conceptuales planteados. Si los modelos conceptuales cubren todas las necesidades que tienen los investigadores, las herramientas desarrolladas cubrirán sus expectativas.

1.5. Estructura de la tesis de máster

La estructura seguida al largo de la tesis de máster es la siguiente:

- En el Capítulo 2 se describe el proceso de negocio del análisis de variaciones genómicas para su caracterización fenotípica. Se detallan formalmente las fases en las que se compone el proceso y se describen los problemas a los que se enfrentan los biólogos durante el desarrollo de dichas fases. Además se ofrece una visión global de la especificación de requisitos del framework desarrollado, así como la estrategia de implementación seguida.
- En el Capítulo 3 se estudian las herramientas de análisis de variaciones, las tecnologías de desarrollo que ofrecen funcionalidad relacionada con el análisis de variaciones y los trabajos académicos que abordan el problema de creación de herramientas de análisis de variaciones genómicas y su caracterización fenotípica.
- En los Capítulos 4, 5 y 6 se describe la formalización cada una de las fases abordadas del proceso: Tratamiento, Alineamiento y Conocimiento. En cada una de ella se describen, los problemas encontrados, las soluciones propuestas, la especificación de requisitos concreta y la implementación llevada a cabo para cada fase.
- Y finalmente, en el Capítulo 7 se presentan las conclusiones del trabajo de investigación de la tesis de máster, las publicaciones asociadas al trabajo realizado y se plantean las nuevas líneas de trabajo futuro.

Capítulo 2

El Proceso de Negocio del Diagnóstico Genético

En este capítulo se describe el proceso de Negocio del Diagnóstico genético llevado a cabo por el Instituto de Medicina Genómica. En este proceso se explican las tareas biólogicas que llevan a cabo para poder ofrecer un diagnóstico genético como soporte para la detección de enfermedades. Así mismo, se describen todas las tareas biólogicas que llevan a cabo para completar este proceso. Para una mejor comprensión de estas tareas se recomienda la lectura del Glosario del Apéndice A.

La colaboración con el Instituto de Medicina Genómica IMeGen, dentro del proyecto Diagen, tiene como objetivo reducir el gap semántico existente entre biólogos e informáticos para desarrollar una herramienta de calidad que de soporte al diagnóstico de enfermedades genéticas. Esta colaboración ha consistido en:

1. Toma de contacto entre los dos grupos de colaboración: La empresa IMeGen explica que tipo de análisis genéticos realizan, una visión general del proceso de negocio que siguen para llevarlos a cabo y las dificultades con las que se encuentran a la hora de desarrollar sus procesos. Nuestro grupo explica un visión general de las lineas de investigación que se siguen en el centro PROS, y cómo los principios de los Sistemas de Información y el Desarrollo Dirigido por Modelos, pueden ayudar en la resolución de las dificultades con las que se encuentra IMeGen al llevar a cabo un análisis genético.

2. Profundizar en el proceso de negocio: IMeGen nos enseña su maquinaria y sus herramientas y nos explican cómo las utilizan para realizar un análisis genético. A partir de esta visita elaboramos una descripción del proceso de negocio.

3. Soluciones a las dificultades del proceso de negocio: Una vez descrito y entendido el proceso de negocio nos explican las dificultades con las que se encuentran a la hora de utilizar las herramientas software y las mejoras que ellos creen que se podrían realizar para mejorar su rendimiento.

4. Lineas de actuación: Se plantean dos lineas de actuación común para solventar las dificultades encontradas.

   a) El desarrollo de un modelo conceptual del genoma humano, que de lugar a un sistema de información de datos genómicos diseñado bajo las buenas prácticas de la Ingeniería del Software y los Sistemas de Información.

   b) El desarrollo de herramientas que extraigan conocimiento del sistema de información genómica basado en el modelo conceptual del genoma humano diseñado. Concretamente, en esta tesis de máster se aborda la creación de un framework, que explota dicho sistema de información genómica, para llevar a cabo un análisis variaciones genéticas para su caracterización fenotípica.

A partir de todas las reuniones celebradas con IMeGen se han elaborado: 1) Diagramas BPMN que describen el proceso de negocio del diagnóstico genético; 2) Diagramas BPMN que describen el proceso de negocio real guiado por las dificultades encontradas; 3) La especificación de requisitos que debería cumplir una herramienta para llevar a cabo un análisis genético; y 4) La aproximación que se va a seguir para desarrollar la herramienta de análisis de variaciones, de manera que cubra sus necesidades.

2.1. El proceso de negocio del diagnóstico genético

En la actualidad, los análisis genéticos que se realizan para la diagnosis de enfermedades son análisis dirigidos, ya que se utilizan únicamente para el soporte a un diagnóstico hecho previamente.

El proceso del diagnóstico genético se inicia cuando un médico detecta en un paciente, en base a la sintomología o a su historial clínico, que puede padecer una enfermedad genética. En ese caso, se toma una muestra de ADN del paciente y se procede a realizar el análisis.

La Figura 2.1 muestra un diagrama BPMN que describe el proceso de diagnosis de enfermedades. En la diagnosis de enfermedades están involucrados tres actores: 1) El paciente que padece o no una enfermedad; 2) El médico que realiza el diagnóstico; y, en el caso de tratarse de una enfermedad genética, 3) El laboratorio genético que comprueba dicho diagnóstico.

El proceso general se inicia cuando un paciente acude a la consulta y explica sus síntomas al médico que le atiende (t1). El médico, en base al conjunto de síntomas indicados, realiza un diagnóstico previo basado en sus conocimientos médicos (t2). Una vez realizado este diagnóstico previo, el médico debe realizar las pruebas médicas correspondientes que le ayuden a cerciorarse de que dicho diagnóstico es correcto. Así mismo, el médico decide qué tipo de pruebas va a realizar (t3): genéticas o no genéticas. En este diagrama no se muestran las enfermedades comunes que no necesitan pruebas médicas como puede ser un resfriado, una indigestión, un golpe leve, etc., ni las subtareas involucradas en las pruebas no genéticas por estar fuera del marco de interés de la presente tesis de máster.

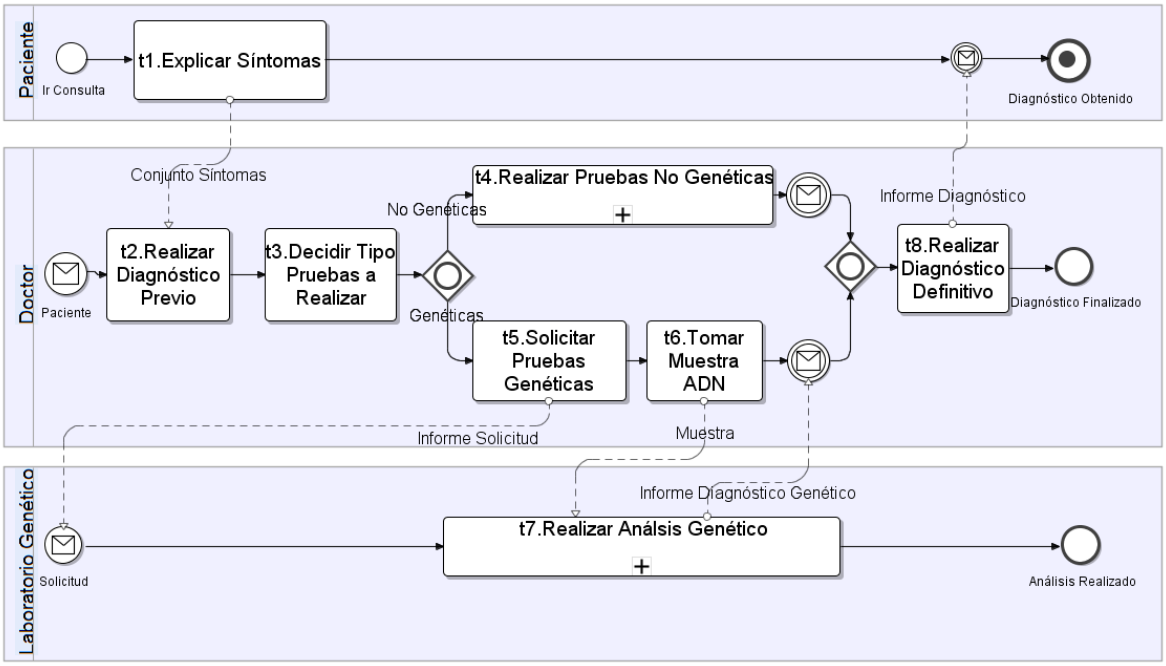


Figura 2.1: El proceso de Diagnosis de Enfermedades

En el caso necesitar una prueba genética, el médico la solicita al laboratorio genético correspondiente (t5), enviando el informe sobre el paciente e indicando su diagnóstico preliminar. El laboratorio genético, en base al informe recibido, procede a realizar un análisis genético de la muestra de ADN tomada por el médico (t6). Una vez realizado el análisis genético (t7), el laboratorio envía el informe de diagnóstico al médico. El médico, a partir del conjunto de pruebas realizadas, genéticas o no genéticas, realiza el diagnóstico definitivo (t8). El proceso finaliza en este momento, es decir, cuando el paciente obtiene el diagnóstico final.

El objetivo de los laboratorios genéticos es efectuar un informe de soporte al diagnóstico en el que se detallen aquellas anomalías que se han detectado en el ADN y que les hace concluir que un paciente es genéticamente potencial o no, a una enfermedad.

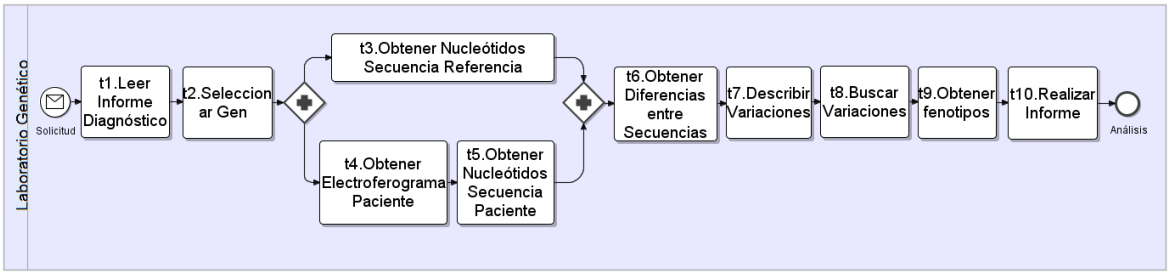


Figura 2.2: El proceso de Análisis Genético

La Figura 2.2 muestra el diagrama BPMN asociado a las tareas que realiza el laboratorio genético para realizar el análisis genético de una muestra de ADN. El proceso de análisis

se inicia cuando el laboratorio genético recibe una solicitud de análisis. En ese momento se procede a leer el informe de diagnóstico recibido (t1) y, según la enfermedad indicada, se selecciona el gen a analizar (t2). Los biólogos son capaces de elegir el gen que es necesario analizar basándose en su experiencia en el dominio. En el marco de esta tesis de máster solo se tendrán en cuenta enfermedades relacionadas con un único gen, descartando aquellas que implican varios genes simultáneamente. Una vez conocido el gen, se obtiene, por un lado, la secuencia de nucleótidos de la referencia del gen seleccionado (t3) y, por otro lado, se secuencia dicho gen a partir de la muestra de ADN del paciente. Las máquinas secuenciadoras obtienen el electroferograma de la secuencia del paciente (t4), que se traduce a nucleótidos (t5), formando una secuencia que representa las moléculas del ADN del paciente. Una vez obtenidas ambas secuencias, se procede a calcular las diferencias entre ellas (t6). Después de esta comparación se procede a describir las diferencias encontradas como variaciones genéticas (t7) y, posteriormente, se buscan las variaciones en distintos repositorios genómicos (t8). Con la información recuperada se obtienen los fenotipos asociados a cada variación (t9). En el marco de esta tesis de máster, el proceso de diagnosis genética se acota teniendo en cuenta solo la relación directa que existe entre una variación genética y una enfermedad, descartando las enfermedades que están provocadas por diversas variaciones. Por ese motivo, se describen y se analizan las diferencias al mismo tiempo, y se descarta el análisis de la interacción entre ellas. Una vez obtenida la caracterización fenotípica de todas las variaciones, se realiza un informe con la información recopilada (t10). El proceso finaliza en este momento, es decir, cuando se ha realizado el análisis genético.

2.2. El proceso de negocio de diagnóstico genético real

Algunas de las tareas descritas en el proceso de negocio del diagnóstico genético no se pueden llevar a cabo de la manera planteada. Esto se debe a un conjunto de limitaciones tecnológicas:

- **Limitaciones de las máquinas secuenciadoras:** La tecnología actual (año 2010) utilizada para la secuenciación genética de ADN tiene ciertas limitaciones que no están relacionadas con los procesos biológicos:
  - Las máquinas secuenciadoras obtienen electroferogramas de corta longitud que se traducen en secuencias de aproximadamente 700 nucleótidos. La cota superior entre diferentes tecnologías se sitúa en los 1000 nucleótidos.
  - Las máquinas secuenciadoras obtienen electroferogramas con los extremos (inicio y final) ilegibles e intraducibles. Como no es posible obtener los nucleótidos de estas regiones, es necesario eliminarlas para llevar a cabo el proceso de traducción a nucleótidos. Esta limpieza de secuencias acorta aún más la longitud de la secuencia, en aproximadamente 50 nucleótidos, dejando la longitud de las secuencias en 650 nucleótidos aproximadamente.
  - Las máquinas secuenciadoras son sensibles al ruido, por esa razón, es posible que se introduzcan señales adicionales. En la traducción a nucleótidos pueden aparecen nucleótidos incorrectos que no reflejan la realidad de la secuencia.

  - Las máquinas secuenciadoras no son capaces de obtener un electroferograma para cada uno de los alelos que forman el ADN (el alelo de la madre y el alelo del padre). Lo que se obtiene es un electroferograma que representa la suma de los dos alelos, por lo que las señales de cada uno de ellos están superpuestas en un único electroferograma. El problema reside cuando se observa, por ejemplo, una posición del electroferograma obtenido. Lo que ocurre es que en este caso no es posible determinar qué señal corresponde a cada uno, ya que no existe una clara separación entre las dos secuencias. Cuando se traduce el electroferograma a nucleótidos, la secuencia que se obtiene es una secuencia representativa, o consenso, de la superposición de ambos alelos. Esta limitación dificulta el análisis de variaciones de manera independiente para cada alelo.

- **Dispersión y heterogéneidad de los Sistemas de Información:** Existe gran diversidad de repositorios que contienen información genómica, donde cada uno de ellos sigue sus propios criterios de estructuración y de representación de la información. La dispersión y heterogeneidad de estos repositorios obliga a abordar la obtención de información siguiendo diferentes metodologías para cada una de las fuentes de datos, lo que supone:

  - Obligatoriedad de conocer cómo y dónde está representada la información que se quiere obtener.
  - Recopilación y análisis de información de manera manual para cada fuente de información.

  La falta de estándares en la manera de representar la información genómica ha provocado una proliferación de diferentes formatos de representación. Estos problemas se han intentado solucionar invirtiendo esfuerzos en aplicaciones e interfaces web de búsqueda, navegación y visualización de información [28], en lugar de invertirlos en estructurar adecuadamente la información genómica que contienen estas fuentes de datos. Este problema justifica la primera línea de investigación planteada: El diseño de un modelo conceptual del genoma humano y la creación de un sistema de información basado en él es necesario para la estructuración de la información genómica.

Todas estas limitaciones tecnológicas influyen en el proceso de negocio. Se observa que el proceso de negocio se ve dificultado por la adición de tareas que pretenden solucionar estos inconvenientes. A las tareas actuales se añaden tareas que no deberían formar parte del proceso de negocio, pero que, por razones tecnológicas, deben estar presentes para poder llevar a cabo todas las acciones requeridas. La aparición de nuevas tareas, ha deformado el proceso de negocio hasta el punto que existe una gran variabilidad en la metodología que cada biólogo sigue para llevar a cabo el análisis genético. Debido a la falta de formalismos que describan el proceso, las tareas no están definidas ni acotadas de manera precisa, ni siguen un orden de ejecución predefinido.

El problema con el que se encuentran los laboratorios genéticos a la hora de desarrollar el proceso de análisis genético descrito es la falta de herramientas que les permita llevarlo a cabo de manera eficiente. Con el objetivo de dar solución a estas limitaciones han proliferado una gran diversidad de herramientas. Sin embargo, aunque muchas de estas herramientas

nuevas son funcionales, la falta de formalismos provoca que cada herramienta solucione solo un subconjunto de los problemas. Por eso, en la actualidad existen herramientas que cubren una tarea, herramientas que cubren solo parte de una tarea, herramientas que cubren un varias tareas o incluso hay tareas para las que no existe ningún tipo de herramienta que la soporte. En resumen, no existe ninguna herramienta que de solución a todas las actividades requeridas [25]. Este problema justifica la segunda línea de investigación planteada: La elaboración de una herramienta que extraiga conocimiento del sistema de información genómica para cubrir todo el proceso de análisis de variaciones.

Las limitaciones mencionadas han afectado principalmente a tres conjuntos de tareas: 1) Tareas relacionadas con la obtención de la secuencia de nucleótidos del paciente; 2) Tareas para la obtención de las diferencias entre la secuencia de referencia y la secuencia del paciente; y 3) Tareas relacionadas con el análisis de cada diferencia encontrada.

El objetivo es formalizar el proceso de análisis genético para establecer una metodología que indique los pasos exactos y el orden de ejecución para llevar a cabo todas las tareas requeridas para realizar un análisis genético.

Para formalizar este proceso se propone una definición y acotación de fases que agrupen tareas de la misma naturaleza que se ejecuten de manera atómica. En otras palabras, tareas que representen funcionalidades concretas y significativas. Como resultado, la formalización propuesta queda dividida en cuatro fases:

1. Fase Tratamiento: Construcción y tratamiento de la secuencia de nucleótidos del paciente. En esta fase se abordan los problemas de las máquinas secuenciadoras (longitud máxima de la secuencia, ilegibilidad en los extremos y errores puntuales) que afectan a la reconstrucción de la secuencia.

2. Fase Alineamiento: Búsqueda de diferencias entre la secuencia de nucleótidos del paciente y la secuencia de nucleótidos de referencia. En esta fase se abordan los problemas de las máquinas secuenciadoras (separación de alelos) que afectan a la identificación de variaciones genéticas en el gen del paciente.

3. Fase Conocimiento: Obtención de la caracterización de un conjunto de diferencias encontradas. En esta fase se abordan los problemas de la heterogeneidad de los repositorios de información genómica que afectan a la obtención de la caracterización fenotípica de las variaciones genéticas.

4. Fase Informe: Generación de un informe que detalle las conclusiones obtenidas. Esta fase queda fuera del marco de esta tesis de máster.

El diagrama BPMN de la figura 2.3 muestra la metodología a seguir una vez formalizado el proceso de análisis de secuencias genómicas. En él se muestra la visión general del proceso de negocio dividida en fases. El proceso de análisis genético consistirá en leer el informe de diagnóstico recibido (t1), seleccionar el gen a analizar (t2), según el gen seleccionado, obtener la secuencia de referencia (t3) y la secuenciación de la muestra del paciente (t4), realizar

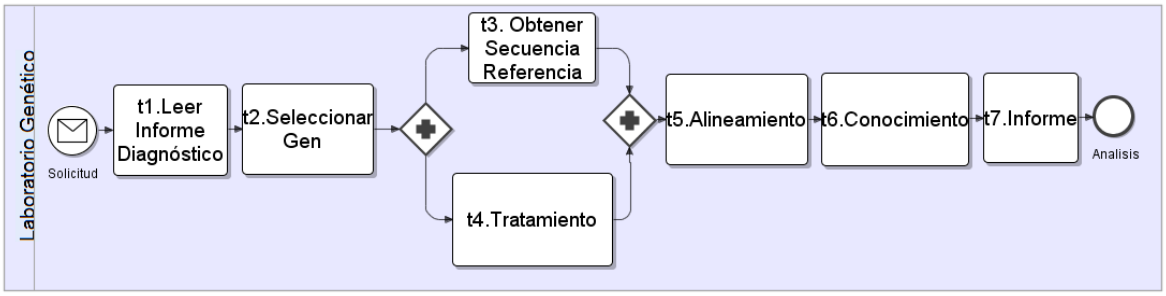


Figura 2.3: Formalización del proceso Análisis Genético

el alineamiento de secuencias en busca de diferencias (t5), obtener el conocimiento de las diferencias localizadas (t6) y generar un informe de diagnóstico (t7).

De esta manera, si el proceso está completamente definido y acotado, será posible readaptarlo ante cualquier mejora tecnológica o incluso ante la aparición de nuevas limitaciones.

2.2.1. Fase de Tratamiento

En la fase de Tratamiento se llevan a cabo las tareas necesarias para la reconstrucción de los nucleótidos de la secuencia de un gen del paciente al que se le está realizando un análisis genético.

Debido a las limitaciones tecnológicas de las máquinas secuenciadoras, se observa que se necesitan ciertas tareas adicionales para completar el objetivo de la fase:

- Los electroferogramas obtenidos tienen longitud limitada: No se puede obtener un electroferograma único que represente toda la secuencia del gen que se quiere secuenciar, y por tanto, tampoco la secuencia de nucleótidos completa de un gen directamente. Para obtener una secuencia de nucleótidos completa se realiza la secuenciación por partes y después se realiza la unión de todos los segmentos secuenciados. Esta limitación añade dos tareas adicionales: 1) Dividir la secuencia en segmentos; y 2) Ensamblar dichos segmentos.
- Los extremos de los electroferogramas obtenidos son ilegibles: No es posible traducir estás regiones a nucleótidos. Para poder obtener los nucleótidos de un electroferograma es necesario desechar esas partes no válidas para la traducción y obtener los nucleótidos a partir del electroferograma restante. Esta limitación añade una tarea adicional: limpiar cada electroferograma secuenciado.
- El electroferograma obtenido puede contener señales no válidas: Estas incorrecciones se propagan en la traducción del electroferograma a nucleotidos. Para poder obtener una secuencia fiable se debe comprobar la corrección de cada nucleótido de la secuencia obtenida. Esta limitación añade una tarea adicional: revisar la secuencia de nucleótidos.

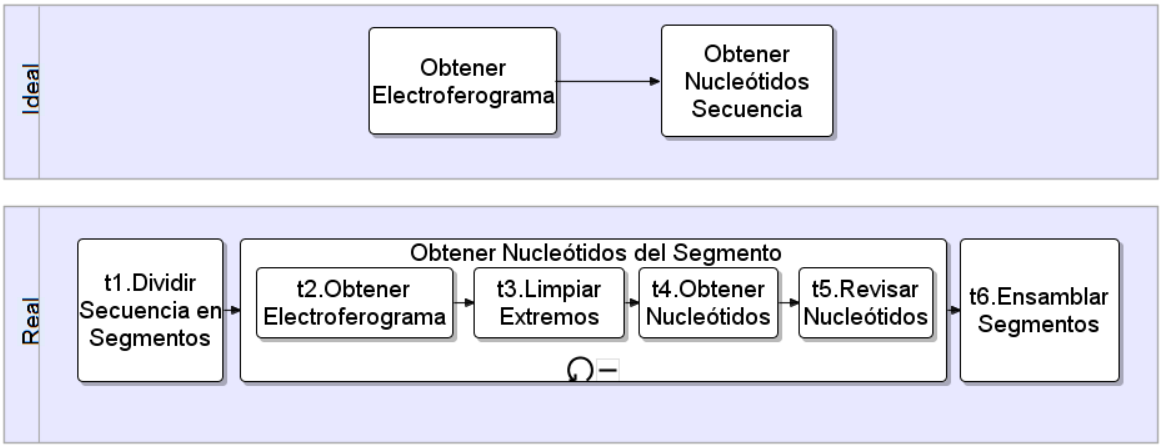


Figura 2.4: Tareas adicionales para obtener la Secuencia del Paciente

Concretamente, tal y como se observa en la Figura 2.4, en la obtención de la secuencia de nucleótidos completa el proceso ideal estaba formado por dos tareas, pero en el proceso real se llevan a cabo seis tareas. En primer lugar, la secuencia se divide en segmentos (t1) y para cada uno de ellos se obtiene la secuencia de nucleótidos (t2-t5). Una vez obtenidas las secuencias de todos los segmentos, se ensamblan para formar la secuencia de nucleótidos del gen completo (t6).

El subproceso para obtener la secuencia de nucleótidos de un segmento está formado por cuatro tareas. En primer lugar, se obtiene el electroferograma del segmento (t2) y, a continuación, se limpian los extremos (t3), que consiste en eliminar aquellas señales del electroferograma ilegibles. Una vez está limpio el electroferograma, entonces es posible realizar la traducción para obtener los nucleótidos (t4). El último paso consiste en revisar la secuencia obtenida nucleótido a nucleótido (t5) para corregir los posibles errores.

La aparición de estas tareas adicionales en el proceso de negocio podrían solucionarse con la evolución de las tecnologías de secuenciación.

2.2.2. Fase de Alineamiento

En la fase de Alineamiento se llevan a cabo las tareas necesarias para obtener las diferencias que existen entre la secuencia de nucleótidos de un gen del paciente y la secuencia de referencia del mismo gen.

Debido a las limitaciones tecnológicas de las máquinas secuenciadoras, se observa que se necesitan tareas adicionales para completar el objetivo de la fase:

- El electroferograma obtenido contiene señales de los dos alelos del gen del paciente: No es posible obtener un electroferograma para cada alelo, y por tanto, no se puede obtener

la secuencia de cada uno de ellos. Para realizar la obtención de las diferencias se debe tener en cuenta que la secuencia que se está analizando es una secuencia consenso que refleja los dos alelos. Esta limitación divide en dos la tarea de obtener las diferencias de la secuencia, es decir, se realizan dos tareas: 1) Búsqueda de diferencias que ocurren en los dos alelos a la vez; y 2) Búsqueda de diferencias que ocurren únicamente en uno de ellos.

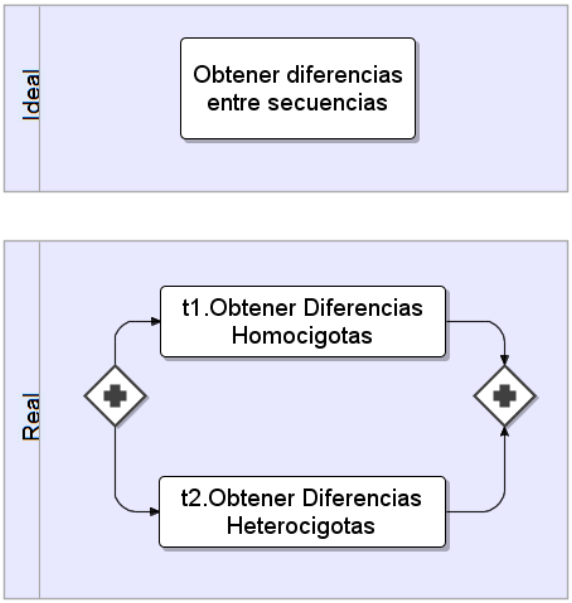


Figura 2.5: Tareas adicionales para obtener las Diferencias entre Secuencias

Concretamente, tal y como se observa en la Figura 2.5, la tarea original de obtener diferencias entre secuencias quedará dividida en dos tareas que se realizan de manera paralela. La primera, para obtener las diferencias homocigotas (t1), cuando existen una diferencia en los dos alelos. La segunda, para obtener las diferencias heterocigotas (t2), cuando existe una diferencia en uno de los dos alelos.

De la misma manera que ocurría en el proceso de negocio de Tratamiento, la aparición de estas tareas adicionales en el proceso de negocio podrían solucionarse con la evolución de las tecnologías de secuenciación. Aunque esta solución queda fuera de la investigación de nuestro dominio, en el campo de la computación se ha investigado la posibilidad de solventarlo desde otro punto de vista. El objetivo planteado consiste en procesar el electroferograma obtenido para, analíticamente, identificar qué señales corresponden a cada alelo. Este problema se denomina inferencia haplotípica [1], y tiene como finalidad obtener, a partir de un electroferograma, dos secuencias de nucleótidos, una que represente a un alelo, y otra que represente el otro alelo.

2.2.3. Fase de Conocimiento

En la fase de Conocimiento se llevan a cabo las tareas necesarias para obtener la caracterización fenotípica de una diferencia encontrada en el gen de un paciente.

Debido a las limitaciones de heterogenidad y dispersión de las fuentes de información, se observa que se necesitan tareas adicionales para llevar a cabo el objetivo de la fase.

- La información genómica sobre variaciones genéticas se encuentra en diferentes fuentes de datos: No es posible realizar una única consulta a una fuente de datos para obtener todo el conocimiento de una variación. Para poder obtener todo el conocimiento es necesario consultar diferentes fuentes de datos. Esta limitación añade dos tareas adicionales: 1) Selección del conjunto de fuentes a analizar y 2) Combinación de todos los datos, una vez consultadas todas las fuentes seleccionadas.
- La descripción de una variación genética, su localización y la interpretación de los resultados obtenidos es diferente para cada fuente de datos: No es posible realizar una descripción de la variación y utilizarla para consultar cada fuente de datos. Tampoco es posible establecer una metodología de búsqueda para localizarla en todas las fuentes ni interpretar los datos obtenidos de la misma manera en todas las fuentes. Para poder obtener información sobre una variación en una fuente concreta es necesario describirla con las variables adecuadas y las coordenadas correspondientes, buscarla utilizando los mecanismos específicos establecidos por la fuente e interpretar los datos obtenidos de forma específica según la estructura de la fuente. Esta limitación origina que las tareas de descripción, búsqueda y caracterización fenotípica se lleven a cabo cada vez, y de manera específica, para cada fuente seleccionada.

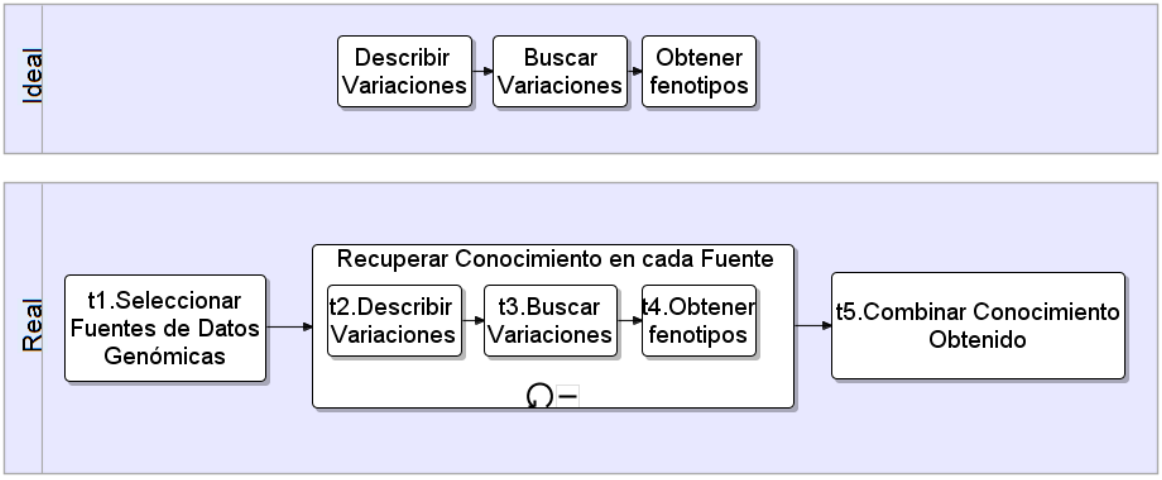


Figura 2.6: Tareas adicionales para el Análisis de Diferencias

Concretamente, tal y como muestra la Figura 2.6, las tareas originales se introducen en un proceso iterativo, precedido de una tarea previa, y seguido de una tarea posterior. En primer

lugar, se realiza una selección de un conjunto de fuentes de datos genómicos donde se van a buscar las variaciones genéticas (t1). En segundo lugar se recupera el conocimiento (t2-t4) para cada fuente seleccionada. Por último se combina todo el conocimiento obtenido, donde quedarán reflejados los fenotipos de las variaciones genéticas obtenidos de diferentes fuentes de datos genómicos.

La aparición de estas tareas adicionales en el proceso de negocio podrían solucionarse con la creación de un sistema de información genómico estructurado y homogéneo. Esta solución es precisamente el punto de partida de esta tesina. En el proyecto Diagen se ha diseñado el Esquema Conceptual del Genoma Humano (CSHG) y se ha creado la Base de Datos del Genoma Humano (HGDB), un sistema de información basado en él. El nuevo sistema de información contiene datos de distintas fuentes que han sido transformados y estructurados en base al modelo conceptual. Gracias al HGDB, es posible la descripción de variaciones de manera homogénea, la búsqueda de variaciones con la misma metodología y la caracterización fenotípica de manera directa.

2.3. Especificación de requisitos: Visión General

Con la finalidad principal de elaborar una herramienta que implemente todo el proceso de negocio, se ha elaborado una especificación de requisitos textual.

- Objetivo: Desarrollo de una herramienta totalmente funcional y robusta que pueda utilizarse en hospitales o centros genéticos como soporte para estudios o diagnósticos de enfermedades genéticas. En la herramienta final se podrán introducir muestras de ADN de un paciente y se obtendrá un informe mostrando las variaciones que tiene la muestra respecto a una secuencia de referencia conocida. En el informe se diferenciaran las variaciones patógenas de las mutaciones genómicas que se ha demostrado pueden ser causantes de una enfermedad, cada una de ellas respaldada con la bibliografía que demuestre el diagnóstico ofrecido.
- Datos de entrada: Conjunto de electroferogramas de la región del paciente que se quiere analizar. Se deben de tener en cuenta:
  - Las regiones que los biólogos seleccionan para la secuenciación pueden variar:
    - Se secuencia un segmento del gen: Los biólogos saben donde está localizada la diferencia o diferencias potenciales que están buscando.
    - Se secuencian varios segmentos disjuntos del gen: Los biólogos no saben donde está localizada la diferencia o diferencias potenciales que están buscando.
  - Cada región se secuencia dos veces en direcciones opuestas: De este modo se obtienen dos versiones de la misma región, lo que permite aumentar la fiabilidad de la secuencia de nucleótidos final.
  - Una vez obtenida la secuencia de nucleótidos a partir del electroferograma, el biólogo revisa que es correcta. Para ello, introducirá un nuevo valor en aquellos nucleótidos que considere incorrectos.

- Datos de salida: Informe que contiene una lista de variaciones genéticas indicando para cada una:

  - Efecto patológico: Clasificación si es una variación asociada a una enfermedad o no.
  - Identificador de la secuencia de referencia utilizada para comparar.
  - Tipo de variación genética: Inserción, Deleción o Indel.
  - Posición de la variación genética en la secuencia de referencia.
  - Valor original del nucleótido o nucleótidos en dicha posición.
  - Valor del nucleótido o nucleótidos del consenso de la muestra donde se ha producido la variación.
  - Exón o exones implicados en la variación.
  - Alcance de la variación: Si la variación se ha producido en los dos alelos (homocigosis) o solo en uno de ellos (heterocigosis).
  - Expresión de la variación encontrada según la nomenclatura de HGVS [12] (nomenclatura estándar para la expresión de Variaciones).
  - Fenotipo asociado
  - Referencia bibliográfica.
  - Fuente bibliográfica.

  El informe generado debe poder guardarse en formato DOC o en PDF.

Una vez conocidos los requisitos generales de la herramienta Diagen. Se elabora el diagrama de contexto y el diagrama de casos de uso del sistema (Figura 2.7). El sistema Diagen interactuará con el biólogo, que es el actor primario que interacciona con el sistema para explotar su funcionalidad. Diagen interactuará, además, con el sistema de información genómica basado en el modelo conceptual del genoma humano, la base de datos HGDB, que es el actor secundario que proporciona soporte al sistema. El sistema está compuesto únicamente por un caso de uso, *Análisis Genético*, ya que la herramienta se ha diseñado, como primera aproximación, únicamente para llevar a cabo esta tarea.

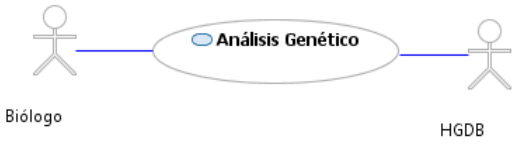


Figura 2.7: Diagrama de Contexto

Análisis Genético

El caso de uso Análisis Genético iniciado por el biólogo, está incluye cinco casos de uso: 1) Seleccionar Gen, donde el biólogo proporciona información; 2) Obtener Secuencias, donde

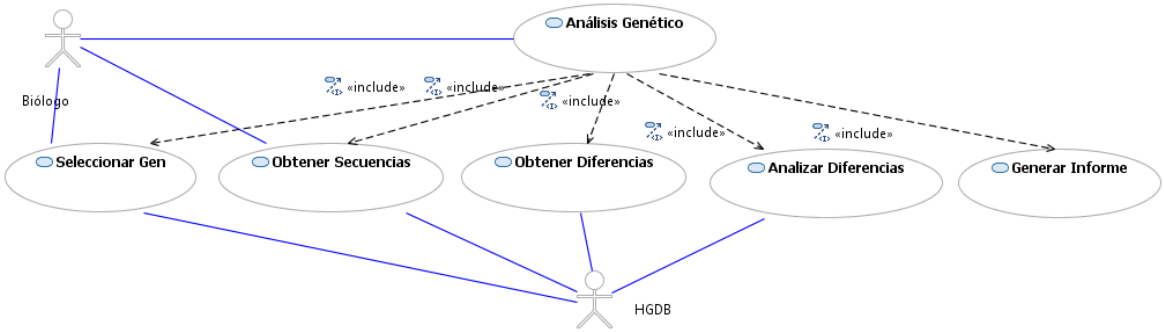


Figura 2.8: Diagrama de Casos de Uso

tanto el biólogo como La HGDB proporcionan información; 3) Obtener Diferencias; 4) Analizar Diferencias, donde es la HGDB la que proporcionan información; y por último, 5) Generar Informe.

- Actores: Biólogo
- Propósito: Realizar un informe de diagnóstico genético a partir de una muestra de ADN.
- Resumen: El biólogo introduce la información genética del paciente y obtiene un informe donde se detalla la caracterización fenotípica (enfermedades) del paciente.
- Postcondiciones:Se ha generado un informe de diagnóstico.
- Casos de uso que incluye: Seleccionar Gen, Obtener Secuencia, Comparar Secuencias, Analizar Diferencias y Generar Informe.

El caso de uso se inicia cuando el biólogo desea realizar un análisis:

1. El sistema pide los datos de entrada:
   a) El biólogo selecciona el gen que desea analizar.
   b) El biólogo introduce la información genética del paciente.
2. El sistema obtiene las secuencias para el análisis.
3. El sistema compara las secuencias obtenidas en busca de diferencias.
4. El sistema analiza las diferencias encontradas.
5. El sistema genera un informe detallado.

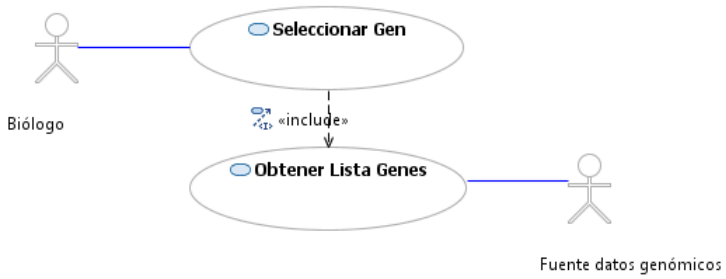


Figura 2.9: Caso Uso Seleccionar Gen

Seleccionar Gen

El caso de uso Seleccionar Gen incluye el caso de uso Obtener Lista Genes (Figura 2.9), ya que el biólogo selecciona el gen del análisis genético a partir de la lista proporcionada por el sistema de información genómica.

- Actores: Biólogo
- Propósito: Elegir el identificador del gen del análisis genético.
- Resumen: El sistema muestra una lista de genes y el biólogo elige el gen del análisis que va a realizar.
- Precondiciones: El biólogo ha indicado en el sistema que va a realizar un análisis.
- Postcondiciones: El gen del análisis ya está definido y no va a cambiar.
- Casos de uso que incluye: Obtener Lista Genes

El caso de uso se inicia cuando el sistema desea conocer el gen del análisis:

1. El sistema proporciona la lista de los genes disponibles:
   a) El biólogo selecciona el gen que desea analizar
   b) El biólogo introduce la información genética del paciente.
2. El sistema establece el gen seleccionado como el gen del análisis durante todo el proceso.

Obtener Lista Genes

El caso de uso Obtener Lista Genes forma parte del caso de uso Seleccionar Gen (Figura 2.9). El sistema proporciona la lista de genes para que el biólogo seleccione el gen del análisis genético.

- Actores: HGDB
- Propósito: Generar una lista de genes.
- Resumen: El sistema obtiene los genes disponibles para el análisis y genera una lista.
- Precondiciones: El biólogo ha indicado en el sistema que va a seleccionar un gen.
- Postcondiciones: Se ha generado una lista de todos los genes disponibles.

El caso de uso se inicia cuando el sistema desea obtener una lista de genes disponibles para el análisis:

1. El sistema consulta los genes disponibles a la HGDB:
   a) La HGDB proporciona una lista con todos los identificadores de los genes almacenados.
2. El sistema crea la lista de genes.

Obtener Secuencias, Obtener Diferencias y Analizar Diferencias

Estos casos de uso se detallan en profundidad en los próximos capítulos.

Obtener Informe

Este caso de uso no se detalla en el marco de esta tesis de máster.

2.4. Implementación: Visión General

El software a desarrollar, obtiene como entrada la secuenciación de ADN de un paciente obtenida por las máquinas secuenciadoras y obtiene como resultado una lista de variaciones, cada una con su fenotipo (indicando si es una enfermedad o no), y su bibliografía asociada.

La finalidad de esta implementación es proporcionar una herramienta funcional que cubra todo el proceso y que descargue al biólogo de tener que preocuparse por implementaciones concretas o por formatos de entrada y salida. La idea no es proporcionar una nueva solución reinventando todo el análisis desde cero y obviando los buenos resultados que otras implementaciones obtienen. Por el contrario, se debe tener en cuenta que ya existen diferentes implementaciones que se utilizan exitosamente. Por lo tanto, para la construcción de dicha herramienta se ha definido el framework Diagen, que se centra en dos puntos clave: 1) Integrar las herramientas externas necesarias (alineadores, algoritmos, bases de datos, etc..) para soportar el proceso

de diagnóstico genético; y 2) Procesar la información que genera cada fase del diagnóstico genético e interconectarla con las siguientes fases del proceso. Este framework es una representación del modelo conceptual CSHG, para dar una implementación del mismo y establecer el vínculo formal entre las distintas herramientas a integrar. De esta forma, mediante la utilización del framework, instanciamos la información de distintas herramientas en un mismo marco conceptual.

Los mecanismos que ofrece el framework puede clasificarse en dos grupos:

1. Integración de funcionalidad ya implementada: Existen algoritmos y herramientas software que desarrollan funcionalidades específicas y que ofrecen resultados precisos aceptados por la comunidad científica. En este sentido el framework, ofrece los componentes software para soportar la traducción de la información de dichas herramientas al modelo conceptual planteado.

2. Proceso de la información: Existe funcionalidad específica para obtener la información necesaria en el proceso del diagnóstico personalizado, que no está implementada o que requiere una serie de mecanismos para procesarla. El framework ofrece una serie de funcionalidad, para refinar esta información y mostrarla de forma más estructurada o conforme a estándares establecidos.

El proceso de análisis de ADN, en el cual se basa el framework, se divide en varias fases, ejecutadas secuencialmente, donde cada una de ellas necesita la información calculada por la fase anterior para realizar su propia tarea. El flujo de datos entre fases no es una tarea trivial debido a la incompatibilidad de formatos entre las salidas de las herramientas de cada una de las fases y las entradas de las herramientas de las fases consecutivas. No existe ninguna formalización ni estándares para definir los resultados que se originan en cada fase. Por ello, conseguir el flujo de datos entre dos fases implica desarrollar un mecanismo de traducción de formatos para cada par de fases interconectadas.

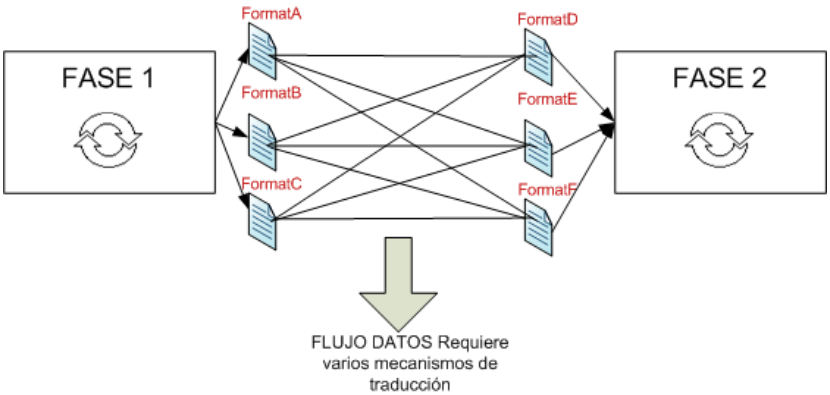


Figura 2.10: Flujo de datos entre fases

De ese modo, para cada una de las herramientas o algoritmos incluidos en una fase, es necesario implementar la traducción para que su formato sea comprensible. En otras palabras, si

cada una de las fases se puede ejecutar con varias herramientas, se tiene que implementar un mecanismo de traducción para cada par de formatos (Figura 2.10). Esta problemática es resuelta por el framework Diagen, que se encarga de realizar la traducción para cada una de las herramientas o algoritmos incluidos en la siguiente fase. En esta tesis de máster se propone que está integración de la información se realice de manera desacoplada, es decir, que no sea necesario que la información siga de manera estricta el proceso establecido por el framework, pudiendo integrarse en la fase que sea necesaria. A continuación se detalla como se realiza dicha interconexión desacoplada.

2.4.1. Interconexión de fases: El Reporte de Resultados

El intercambio de información entre dos fases supone que ambas comparten la misma semántica de datos, es decir, existen conceptos comunes que se comparten en ambas. Por eso, en lugar de invertir esfuerzos en desarrollar traductores acoplados a las herramientas o proponer un estándar de bajo nivel, como un formato textual, se propone definir un modelo conceptual común entre cada par de fases interconectadas que guíe el flujo de datos entre dichas fases. Un modelo conceptual común que modela los conceptos que representan los resultados obtenidos después de la ejecución de la fase (que también representan la información de entrada para la fase siguiente). A su vez el framework es una implementación de dicho modelo conceptual, estableciendo un formalismo común que permite tanto la integración como el proceso de la información.

Gracias a la definición de un modelo conceptual común que interconecta cada una de las fases, ya no es necesario implementar un mecanismo de traducción para cada par de formatos. Para cada una de las herramientas integradas bastará con la implementación de un mecanismo de traducción que exprese sus resultados en términos del modelo conceptual (Figura 2.11):

1. Integración de funcionalidad ya implementada: Se implementa un mecanismo de traducción, que mediante un módulo Input y un módulo Output interactúa con la herramienta. La información de entrada expresada mediante el modelo conceptual de entrada se transforma al formato específico de la herramienta (módulo Input), y la información de salida expresada en el formato específico de salida se transforma al modelo conceptual de salida (módulo Output).

2. Proceso de la información: La funcionalidad desarrollada en el contexto de Diagen se basa en el modelo conceptual del framework. Por esa razón no es necesario implementar ningún mecanismo de traducción adicional. Es posible instanciar la información utilizando las mismas clases que utiliza el framework.

Cabe destacar que utilizando esta aproximación, cada uno de los componentes del framework que implementa una fase, son independientes entre si. Es decir, es posible utilizar la funcionalidad que proporciona el framework para la fase de alineamiento utilizando una herramienta de tratamiento de muestras totalmente ajena al mismo. De esta forma, se ofrece unconjunto

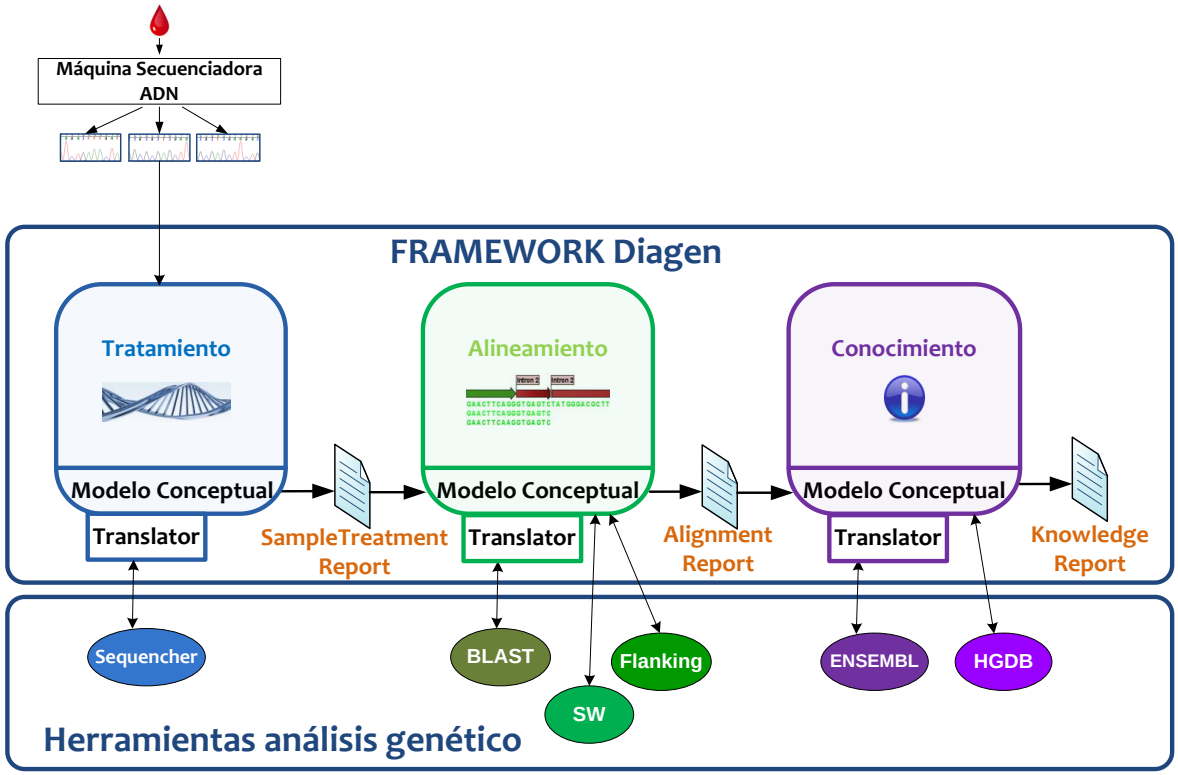


Figura 2.11: El framework Diagen

de componentes modulares que pueden ser acoplados, mediante los mecanismos de traducción oportunos, en otros procesos de negocio diferentes al presentado en esta tesis de máster.

Según las ideas introducidas, se diseña un modelo conceptual para cada par de fases interconectadas del proceso de análisis de variaciones de tal forma que, en una fase representa la salida de información y la otra representa la entrada. Sin embargo, aunque este modelo conceptual es común entre las dos fases que interconecta, expresa los conceptos que resultan de la fase precursora. Por ese motivo cada uno de ellos se llama Reporte de Resultados o Report, porqué ofrece un resumen de los resultados que se han obtenido después de la ejecución de la fase, quedando así todos los conceptos completamente acotados y definidos. Cada Reporte de Resultados se utiliza para guiar el flujo de datos entre fases consecutivas, flujo que se consigue obteniendo las salidas de la fase previa y preparando las entradas para la fase siguiente.

Siguiendo esta estrategia de implementación es sencillo agregar funcionalidad en una fase, ya que, si se trata de funcionalidad ya implementada, únicamente es necesario implementar un mecanismo de traducción en términos del modelo conceptual. El inconveniente que presentan los transformadores textuales entre herramientas, es que debe tener en cuenta la implementación interna de cada par de herramientas que se quiere interconectar y establecer las relaciones conceptuales entre ambas. En el framework, sin embargo, mediante el uso del modelo conceptual como guía de interconexión, los transformadores implementados son transformadores meramente sintácticos, ya que el modelo conceptual representa la semántica de

la información a integrar. Esta ventaja es consecuencia del hecho que el modelo conceptual haya sido definido teniendo en cuenta, de manera exhaustiva el dominio genómico, mediante la colaboración con bioinformáticos y un extenso estado del arte de las herramientas que se utilizan. Por lo tanto, se garantizá el soporte a un amplio número de herramientas y conceptos. La ventaja fundamental del modelo conceptual es que no es necesario conocer el funcionamiento interno de cada herramienta para la implementación de todos los transformadores, tan solo sus resultados.

Capítulo 3

Estado del Arte

A la hora de llevar a cabo un análisis genético de ADN para obtener variaciones en la muestra del paciente y su posterior caracterización fenotípica, los biólogos utilizan y combinan un conjunto de herramientas diferentes.

Los biólogos afirman que, hasta el momento, no se ha desarrollado ninguna herramienta que reciba como entrada una muestra de ADN y se obtenga como salida un informe en el que se indiquen la lista de variaciones, las enfermedades asociadas y las referencias bibliográficas que respalden las asociaciones entre genotipo-fenotipo.

En este capítulo se ha realizado una revisión de un conjunto de herramientas para la ejecución de análisis de secuencias de ADN. El objetivo de esta revisión es comprender porqué las herramientas analizadas no cubren las expectativas de los biólogos que las utilizan.

Con el objetivo de desarrollar herramientas más funcionales que las existentes, surge una nueva iniciativa para el diseño de herramientas bioinformáticas. La idea consiste en proporcionar implementaciones de tareas específicas del proceso de análisis y dejar en manos del biólogo la selección de tareas que quiere incluir en su herramienta. Esta funcionalidad específica se ofrece en forma de frameworks, kits de herramientas, librerías y también servicios web. Estos entornos implementan tareas como por ejemplo: conversión de formatos de secuencias, algoritmos de alineamiento, traslación de nucleotidos a aminoácidos, etc. De esta manera, construir una nueva herramienta consiste en integrar la funcionalidad disponible y ofrecerla al usuario en un entorno único.

Sin embargo, estas herramientas son resultado de proyectos en los que se limitan a integrar funcionalidad, por lo que su aceptación entre los biólogos es limitada debido a que en ocasiones estos las encuentran difíciles de instalar, configurar y extender [30].

El problema de insatisfacción de los biólogos con las herramientas de análisis persiste, por lo que diferentes trabajos en el contexto académico abordan el problema mediante la aplicación de los principios de la ingeniería del software. El objetivo de todas ellas es desarrollar herramientas de calidad que se ajusten de manera precisa a los requisitos de los usuarios.

3.1. Herramientas de análisis de variaciones genómicas

Con el fin de capturar los problemas que presentan las herramientas de análisis de ADN se han analizado un conjunto de herramientas de entre las más relevantes en el dominio [26, 13]: Sequencher [9], SeqScape [7], Mutation Surveyor [29], Codon Code Aligner [11], Polyphred [6], InSNP [21] y la herramienta Web BLAST de NCBI.

El análisis se ha llevado a cabo utilizando muestras reales del gen BRCA1 proporcionadas por el laboratorio genético IMEGEN, a fin de proporcionar al análisis resultados más precisos.

El objetivo de este análisis es determinar las ventajas y desventajas que ofrece cada una de las herramientas, así como identificar que partes del proceso de análisis de secuencias de ADN cubren y en qué medida proporcionan resultados adecuados. Otra característica importante a analizar es la forma en que cada una de las herramientas ofrece sus resultados para ser fácilmente integrados junto con otras herramientas.

Para cada una de las herramientas se analizan:

1. Tareas de la fase de Tratamiento:
   - *a*) Limpieza de secuencias: Comprobar si soporta la limpieza de secuencia que elimine los segmentos de las secuencias bajo ciertos valores de calidad.
   - *b*) Edición de secuencias: Comprobar si soporta la edición del valor de un nucleótido de la secuencia obtenida. Observar de qué grado de la libertad dispone un usuario para modificar el valor (qué valores puede introducir y cuales no), así como el efecto que esta acción tiene en las muestras.
   - *c*) Ensamblaje: Comprobar si soporta la identificación de la posición de un fragmento de la secuenciación respecto a la secuencia de referencia. Observar si permite limpiar los fragmentos y ajustar parámetros de configuración con el fin de obtener resultados más precisos en la localización.
2. Tareas de la fase de Alineamiento:
   - *a*) Variaciones buscadas: Comprobar si soporta la localización de variaciones. Observar que variaciones intenta localizar así como cuales es capaz de encontrar y cuales no: Inserciones, deleciones y sustituciones en homocigosis y en heterocigosis.
3. Tareas de la fase de Conocimiento:
   - *a*) Obtención del fenotipo: Comprobar si soporta la identificación del efecto de la variación localizada.
   - *b*) Obtención de la bibliografía: Comprobar si soporta la búsqueda bibliográfica de referencias que identifiquen la variación y la asocien con un fenotipo concreto.

La estrategia seguida para el llevar a cabo el análisis ha sido:

1. Instalación de las herramientas: Las herramientas Sequencher, SeqScape, Mutation Surveyor, Codon Code Alignment y InSNP se han instalado bajo el sistema operativo Windows 7. Sin embargo, la herramienta Polyphred, únicamente soportada en Linux, se ha instalado bajo Ubuntu v8.04. La mayoría de las herramientas analizadas solo estaban disponibles en su version de prueba, donde la funcionalidad proporcionada era limitada (subconjunto de la funcionalidad total).

2. Lectura de los tutoriales, guias de usuario y manuales de las herramientas: Con el fin de entender los principios generales de las herramientas, conocer la estructuración de la interfaz que proporcionan y la funcionalidad que en principio proporcionan.

3. Ejecución de la funcionalidad: Se utilizan las muestras proporcionadas por el laboratorio IMEGEN para probar toda la funcionalidad indicada en la documentación de cada herramienta.

    a) Creación del proyecto para el análisis: Se importan los archivos que contienen las muestras dentro de un proyecto común. A continuación se configuran los parámetros de análisis.

    b) Recomposición de la secuencia del paciente: A partir de los archivos proporcionados se realiza la composición de la secuencia del paciente mediante la secuencia de referencia.

    c) Comparación entre secuencias: Se realiza el alineamiento y se buscan las variaciones entre ambas secuencias.

    d) Análisis de las variaciones encontradas: Se comprueba que diagnóstico ofrecen para cada una de las variaciones que han localizado.

    e) Generación de informes: Se generael informe final y se exporta en los formatos soportados.

4. Comparación de resultados: Se comprueban los resultados obtenidos por cada herramienta.

Una vez analizadas las herramientas se concluye que ninguna de ellas tiene soporte para todas las fases del proceso de análisis, ya que las herramientas Sequencher, SeqScape, Codon Code Aligner y InSNP (esta última parcialmente) dan soporte a las fases de Tratamiento y Alineamiento, y las herramientas Mutation Surveyor y Polyphred únicamente a la fase de Alineamiento. Sin embargo, ninguna de ellas ofrece la posibilidad de obtener el Conocimiento de las variaciones ofreciendo el fenotipo y la bibliografía asociada a cada variación.

Las herramientas actuales requieren soporte adicional para completar todas las actividades de la fase de Tratamiento, ya que por ejemplo algunas carecen de soporte para la limpieza de secuencias o la edición de nucleótidos. Lo mismo ocurre en la fase de Alineamiento, ya necesitan mejorar la localización de variaciones. Por ejemplo, la localización de variaciones en heterocigosis no está satisfactoriamente resulta en ninguna de las herramientas.

Además todas ellas requieren soporte adicional para llevar a cabo la fase de Conocimiento. Algunas herramientas, una vez localizadas las variaciones, ofrecen al usuario la posiblidad de

	Sequencher	SeqScape	CodonCode Aligner	Mutation Surveyor	Polyphred	InSNP
Secuenciación						
Limpieza secuencias	√	√	√	×	×	√
Edición de nucleotidos	√	√	√	×	×	×
Ensamblaje	√	√	√	√	√	×
Alineamiento						
Homocigosis						
Inserciones	√	√	√	√	×	×
Deleciones	√	√	√	√	×	×
Sustituciones	√	√	√	√	√ (solo SNP)	√ (solo SNP)
Heterozygosis						
Inserciones	×	Parcialmente	Parcialmente	Parcialmente	×	Solo posición inicial
Deleciones	×	Parcialmente	Parcialmente	Parcialmente	×	Solo posición inicial
Sustituciones	√	√	√	√	√ (solo SNP)	√ (solo SNP)
Conocimiento						
Identificación fenotipo	×	√(añadiendo bd)	×	√ (añadiendo bd)	×	×
Identificación bibliografía	×	×	×	×	×	×

Cuadro 3.1: Comparación herramientas

crear una base de datos local para almacenar variaciones y sus descripciones. De esta manera la herramienta es capaz de localizar la variacion en la base de datos local. Sin embargo, esta solución, además de no estar completamente clara en las herramientas, no es viable ya que supone que los usuarios de las herramientas realizen la integración de las diferentes bases de datos de variaciones manualmente.

3.2. Entornos de desarrollo de software bioinformático

Existen diversidad de proyectos y grupos de trabajos dedicados a la implementación de software y herramientas bioinformáticas. Estas iniciativas proveen entornos o herramientas que llevan a cabo actividades relacionadas con el análisis genético de ADN. Algunos de estos ejemplos son:

BioPerl [31]: Es un framework de código abierto para el desarrollo de aplicaciones en lenguaje Perl. Proporciona objetos reusables para representar y gestionar tipos de datos biológicos, principalmente relacionados con las secuencias genéticas. Entre la funcionalidad más destacada se encuentra:

- Descripción, indexación, transformación y anotación de secuencias.
- Alineamientos entre secuencias
- Soporte para búsquedas en secuencias genéticas y trasformación y manipulación del formato de resultados.
- Primitivas de análisis de secuencias como por ejemplo Pattern Matching.
- Wrappers para el acceso a otras herramientas o algoritmos, como por ejemplo Blast.

Algorithm 3.1 Ejemplo BioPerl

```
use Bio::DB::EMBL;
use Bio::SeqIO;

my $db= new Bio::DB::EMBL();
my $seq=$db->get_Seq_by_acc("U14680);
my $seqout=new Bio::SeqIO(-format=> "genbank");
if(defined $seq){
      $seqout->write_seq($seq);
}
```

- Soporte para respresentación 3D e información estructural sobre las proteínas..

El Algoritmo 3.1 muestra un ejemplo en el que se accede a la base de datos de EMBL para recuperar una secuencia mediante el identificador "U14680). Posteriormente esta secuencia se transforma al formato de Genebank.

BioPython [10]: Es un framework de código abierto para el desarrollo de aplicaciones en lenguaje Python. Proporciona una colección de aproximadamente 30 clases cuya funcionalidad más destacada es:

- Interacción con Blast, FastA y ClustalW de manera local o remota.
- Acceso a recursos bibliográficos como PubMed.
- Traducción entre multiples formatos de bases de datos biólogicas.
- Primitivas de análisis de secuencias como por ejemplo Pattern Matching

Algorithm 3.2 Ejemplo BioPython

```
>>> from Bio.Seq import reverse_complement, transcribe,
back_transcribe, translate
>>> my_string = "GCTGTTATGGGTCGTTGG"
>>> reverse_complement(my_string)
'CCAACGACCCATAACAGC'
>>> translate(my_string)
'AVMGRW'
```

El Algoritmo 3.2 muestra como obtener el complementario de una secuencia genética (mediante el método reverse_complement()) y obtener la traducción de una secuencia genética a proteinas (mediante el método translate()).

BioJava [18]: Es un framework de código abierto para el desarrollo de aplicaciones en lenguaje Java. Proporciona métodos para el análisis de secuencias, herramientas para la conversión de formatos entre herramientas, librerias para la manipulación de secuencias, etc. Para ello, se definen un conjunto de clases tales como "Secuencia", "Alineamiento", "Anotaciones", "Alfabeto", "Simbolo", "BaseDatos", etc. y un conjunto de métodos como por ejemplo: representar gráficamente una secuencia, acceder a una base de datos genómica, convertir una secuencia de un formato a otro, traducir los resultados de una herramienta de análisis a otra, etc.

Algorithm 3.3 Ejemplo Biojava

```
import org.biojava.bio.symbol.*;
import org.biojava.bio.seq.*;

public class Translate {

  public static void main(String[] args) {
    try {
      //create a DNA SymbolList
      SymbolList symL = DNATools.createDNA("atggccattgaatga");

      symL = RNATools.transcribe(symL); //transcribe to RNA
      symL = DNATools.toRNA(symL);//transcribe to RNA
      symL = RNATools.translate(symL);//translate to protein
   }
}
```

El algoritmo 3.3 muestra un ejemplo de la funcionalidad de Biojava. Mediante la importación de "org.biojava.bio.*" es posible programar una clase *Translate* que utiliza los objetos de biojava, en este caso *SymbolList*, para realizar la transcripción de ADN a ARN (mediante los metodos *transcribe*() y *toRNA*()) o para la traducción de ADN a proteínas (mediante el método *translate*()).

SeqAn[14] Es un framework escrito en C++ que proporcionan estructuras de datos e implementaciones de algortimos para la implementación de herramientas bioinformáticas para el análisis de secuencias genéticas. SeqAn proporciona tipos de datos como por ejemplo: ADN, CodigosUIPAC, Aminoácidos, etc. y métodos como Fasta(), para convertir un fichero en formato FASTA, SmithWatterman() para ejecutar una implementación del algoritmo de alineamiento, métodos para parsear un reporte obtenido por el algoritmo Blast, etc.

Existen multiples framworks, librerias y servicios web adicionales para el desarrollo de aplicaciones software. Como por ejemplo: NCBI C++ Toolkit [34], Sequence Class Library[33], EBI Web Services [19] o Entrez Utilities (http://eutils.ncbi.nlm.nih.gov/) entre otros.

3.2.1. Ejemplos de herramientas

Bioclipse [30] Es un workbench (Figura 3.1) para el análisis de recursos biológicos y químicos, tales como moléculas, proteinas, secuencias, espectros, etc. Proporciona edición,

Algorithm 3.4 Ejemplo SeqAn

```
#include <seqan/align.h>
using namespace seqan;
int main()
{
    Align< String<char> > ali;
    appendValue(rows(ali), "aaactgtgctgcaagtc");
    appendValue(rows(ali), "aaactgttgcaagtc");

    localAlignment(ali, Score<int>(3,-3,-2, -2), SmithWaterman());
}
```

visualización, conversión de formatos y análisis biológicos y químicos. Está escrito en Java y permite la extensión de funcionalidad mediante pluggins. Mediante la definición de workflows se encarga de componer funcionalidad para el análisis genético y bioquímico del ADN. Para la gestión de secuencias (manipulacion de secuencias, conversión de ficheros, rutinas de análisis) utiliza el framework Biojava. Para el alineamiento de secuencias utiliza servicios Web como por ejemplo: KAlign.

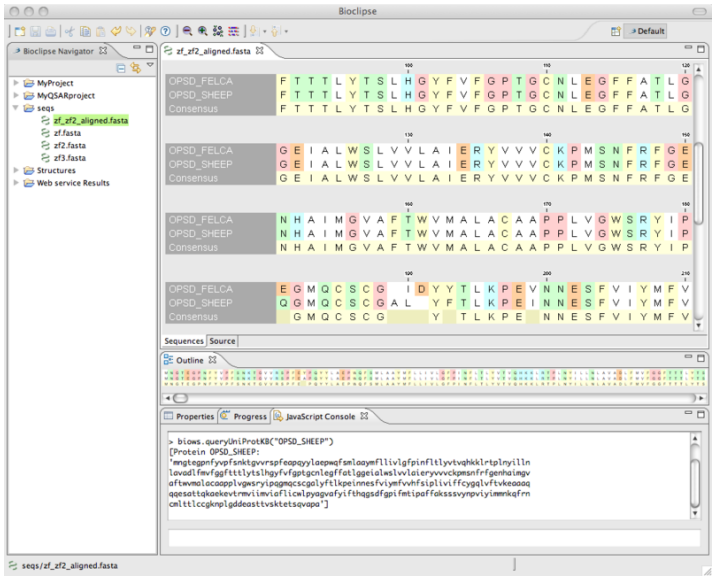


Figura 3.1: Interfaz Bioclipse

ELXR [27] Es un software (Figura 3.2) para automátizar el proceso manual de diseño de primers para la secuenciación de PCR. Realiza operaciones utilizando secuencias de ARN para la obtención de los nucleótidos necesarios para la secuenciación de exones y sus regiones adyacentes. La finalidad del software es obtener las secuencias necesarias para la preparación de la secuenciación con el objetivo de realizar la detección de variaciones genéticas. El código

fuente se ha desarrollado con el lenguaje Perl y se han utilizado varios módulos de BioPerl, como por ejemplo el módulo CGI para el desarrollo de la aplicación Web.

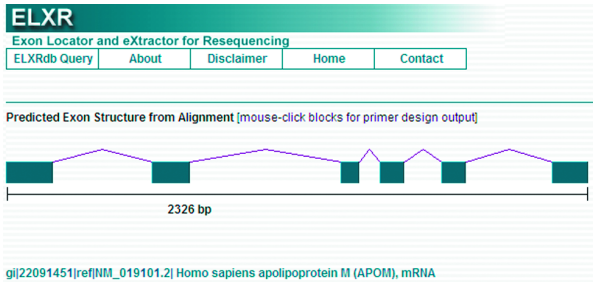


Figura 3.2: Interfaz ELXR

Existen multiples implementaciones que utilizan los entornos de desarrollo biólogicos disponibles para el desarrollo de herramientas de análisis de ADN. Estas herramientas, aunque consiguen ser funcionales, muchas de ellas son soluciones de integración de funcionalidad, y en muchas ocasiones en lugar de proveer nueva funcionalidad de manera intuitiva mediante interfaces sencillas, son dificiles de instalar, de configurar y sobretodo de extender. El problema reside en que la composición de tareas se implementa a bajo nivel, es decir, integrando tareas muy específicas y teniendo en cuenta aspectos de implementación, como por ejemplo formatos de entrada y salida.

Aquellas herramientas que consiguen solventar estas dificultades y se convierten en herramientas útiles y sencillas deben invertir muchos esfuerzos en solventar los problemas de bajo nivel que estos entornos de desarrollo presentan, como por ejemplo la interacción entre diferentes herramientas o la traducción de formatos textuales entre módulos.

3.3. Trabajos académicos que abordan el desarrollo de herramientas biológicas mediante modelos conceptuales

Pierre: Desarrollo de interfaces de usuario biólogicas mediante MDA [15] Pierre es un framework para la generación parcial de interfaces de usuario para la búsqueda y navegación en repositorios de datos biológicos (Figura 3.3) mediante una aproximación dirigida por modelos. Llevan a cabo la identificación de requisitos recurrentes en las interfaces y la adopción de la arquitectura dirigida por modelos para soportarlos. Plantean el desarrollo de repositorios en base a la especificación de servicios. Pierre permite a los desarrolladores de interfaces personalizarlas con funcionalidad adicional mediante la implementación de servicios como software pluggins.

Memops (The MEta-MOdel Programing System)[24]: MEMOPS es un framework para el modelado en UML y la generación automática de código para el desarrollo de aplicaciones para el almacenamiento y recuperación de información biólogica. En la Figura 3.4 se

Figura 3.3: Interfaz Pierre

muestra como grácias a la arquitectura Memops se puede desarrollar un modelo de datos del domino biólogico que se quiere gestionar. Este se estructura en paquetes, y mediante la generación automática de código se elaboran APIs, para el acceso a la información, y esquemas, para el almacenamiento de la información.

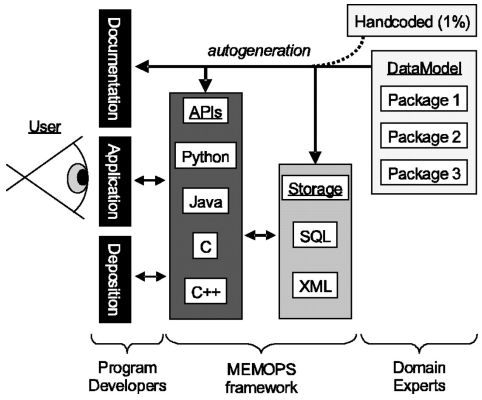


Figura 3.4: Arquitectura MEMOPS

Diferentes trabajos se han aplicado utilizando modelado conceptual para el desarrollo de herramientas biólogicas. Entre ellos, encontramos trabajos que abordan el diseño de interfaces usables para que se ajusten a las necesidades de los biólogos y el diseño de herramientas para la recuperación de información. Estas aproximaciones consiguen buenos resultados y obtienen interfaces usables y amigables para el usuario. Sin embargo, estos trabajos académicos unicamente se han dirigido hacia la navegación y búsqueda en repositorios y no en el análisis de datos con la finalidad de obtener conocimiento adicional, como por ejemplo, para obtener una lista de fenotipos para el soporte al diagnóstico.

Capítulo 4

Fase Tratamiento

La fase Tratamiento es la fase en la que los archivos obtenidos por las máquinas secuenciadoras de ADN se recomponen con el fin de reconstruir una secuencia genética que pueda analizarse mediante las diferentes herramientas bioinformáticas de análisis genético.

En primer lugar se proporciona un conjunto de conceptos biológicos para comprender el proceso de secuenciación y las tareas necesarias para la reconstrucción de la secuencia genética.

En segundo lugar se identifican los problemas encontrados a la hora de reconstruir la secuencia y las soluciones que se adoptan para solventarlas. En el diseño de esta fase se abordan las siguientes limitaciones de las máquinas secuenciadoras: 1) La corta longitud de los electroferogramas obtenidos; 2) La necesidad de limpiar los extremos de los electroferogramas; y 3) La presencia de ruido en los electroferogramas.

En tercer lugar se plantean los diferentes casos de uso necesarios para la reconstrucción de la secuencia genética a partir de la información proporcionada por los biólogos. Esta información puede provenir directamente de las máquinas secuenciadoras, por lo que se deben tener en cuenta todas las limitaciones detectadas, o bien puede ser información previamente procesada.

En cuarto lugar se explica el modelo conceptual de desarrollo y su correspondencia con el CSHG. Se presentan las entidades conceptuales y los métodos necesarios para llevar a cabo todos los casos de uso para la reconstrucción de la secuencia genética.

Por último se explica la implementación llevada a cabo y el modelo conceptual de Reporte de Resultados SampleTreatmentReport, utilizado para expresar los resultados de la fase de Tratamiento de manera precisa.

4.1. Background Biológico

En este apartado se introducen un conjunto de conceptos biológicos para ayudar a comprender el proceso de secuenciación y las limitaciones de las máquinas secuenciadoras.

La Información Genética

Analizar el ADN de un individuo mediante el uso de herramientas software o algoritmos es posible mediante la expresión del ADN en un formato textual. Concretamente, la cadena de ADN se expresa mediante una cadena de caracteres, que se denomina secuencia, y cada uno de los caracteres representan los nucleótidos del ADN del individuo. El caracter A representa la molécula de Adenina, el caracter C representa la molécula de Citosina, el caracter G representa la molécula de Guanina y el caracter T la molécula de Timina.

La secuenciación

Los biólogos seleccionan una región del ADN a secuenciar y las máquinas secuenciadoras proporcionan la representación del ADN en forma de electroferograma. Un electroferograma es una representación gráfica de las señales obtenidas por una técnica de secuenciación llamada electroforesis [5]. Un electroferograma (Figura 4.1) es una sucesión de señales de diferentes colores, donde cada uno de los colores representa un tipo de molécula de ADN, Adenina, Citosina, Guanina y Timina.

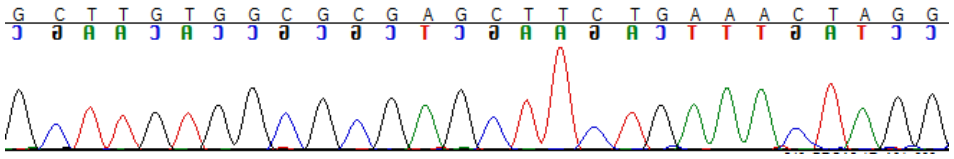


Figura 4.1: Electroferograma

Un electroferograma obtenido con las tecnologías de secuenciación actuales tiene aproximadamente 700 nucleótidos, con los extremos ilegibles y con algunos errores puntuales introducido por las máquinas secuenciadoras.

La Secuencia de Referencia

Para cada gen del genoma, existe una secuencia fictícia, llamada secuencia de referencia, que se utiliza para llevar a cabo algunas operaciones de análisis. Esta secuencia de referencia representa la secuencia de nucleótidos de todos los seres humanos, y se dice que es ficticia porqué está hecha a partir de un estudio realizado a un conjunto de individuos. Esta secuencia de referencia se utiliza en la fase de Tratamiento como base para la reconstrucción de la secuencia genética del individuo secuenciado.

Transcripción del ADN y Unidad de Transcripción

El ADN está formado por un conjunto de genes que son regiones de nucleótidos responsables de una función celular concreta. Estos genes, a su vez contienen dos tipos de regiones: exones e

intrones. Los exones están formados por aquellos nucleótidos que intervienen en la creación de proteínas para la función del gen, por lo que se denominan regiones codificantes. En cambio, los intrones están formados por los que no tienen ningún efecto en ella, por lo que se denominan regiones no codificantes.

El proceso de transcripción es el proceso que transfiere la información codificante del ADN para la formación de proteínas. En el proceso de transcripción se descartan los intrones, y los exones se agrupan para formar lo que denominamos ARN (Figura 4.2). El ARN mensajero, sale del núcleo celular y así se codifican los aminoácidos que forman las proteínas responsables de la función del gen.

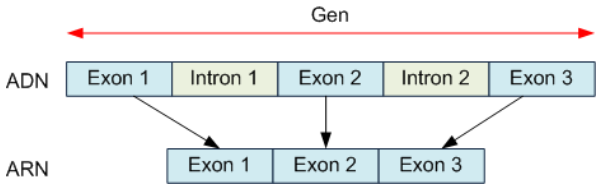


Figura 4.2: Formación del ARN

Una unidad de transcripción es el conjunto de índices, respecto a la secuencia de referencia, que indican el principio y el final de cada uno de los exones que forman el ARN.

4.2. Problemas detectados y Soluciones

A fin de proporcionar una secuencia genética que se pueda analizar en las fases siguientes del proceso de análisis es necesario realizar un preproceso del contenido proporcionado por las máquinas secuenciadoras. De esta manera se consigue una secuencia genética correcta y completa. Para ello la solución que los biólogos adoptan para cada uno de los problemas detectados es la siguiente:

- Longitudes cortas de los electroferogramas: Las máquinas secuenciadoras no son capaces de proporcionar un electroferograma completo, por lo que no se puede obtener una secuencia completa de un gen en una sola iteración de secuenciación. Como solución, se propone secuenciar el ADN por segmentos y posteriormente realizar un ensamblado.
- Extremos ilegibles en los electroferogramas: Las máquinas secuenciadoras ofrecen una señal ilegible en el electroferograma de secuenciación al inicio, cuando empiezan a secuenciar, y al final, cuando no son capaces de seguir con la secuenciación. Como solución, se propone realizar un corte en cada extremo, desechando las partes ilegibles.
- Ruido introducido en los electroferogramas: Las máquinas secuenciadoras no son 100 % fiables. En ocasiones se producen interferencias o ciertas combinaciones de nucleótidos introducen errores puntuales en la secuenciación. Como solución, se propone la revisión manual de los nucleótidos de cada segmento secuenciado.

4.2.1. Ensamblaje de secuencias

Debido a que no es posible secuenciar todo el genoma simultáneamente y obtener un único electroferograma, éste se secuencia por trozos, denominados contigs o segmentos, y posteriormente se realiza un ensamblado de todos ellos.

El proceso de secuenciación de un segmento es un proceso costoso que se realiza manualmente. Obtener una secuencia textual completa implica secuenciar, limpiar y corregir una gran cantidad de segmentos. Este es un proceso muy costo que no aporta la suficiente información como para compensar el esfuerzo y tiempo empleado. Por esa razón, normalmente se selecciona la región del genoma de interés y se secuencia lo necesario.

En aquellos casos en los que el objetivo de la secuenciación es realizar un pequeño análisis de ADN, únicamente se secuencia lo imprescindible. Los laboratorios no necesitan conocer el genoma completo, así que, las regiones que se suelen seleccionar para secuenciación son unidades más pequeñas del genoma: los genes o, incluso, los exones de un gen.

Para llevar a cabo los análisis de búsqueda de variaciones, como los que se consideran en el marco de esta tesis de máster, se secuencian los exones y se prescinde de los intrones. La razón por la que se prescinde de estas regiones es porque las variaciones que ocurren en los intrones no afectan (aunque existen excepciones) en la creación de proteínas, y por lo tanto, en la aparición de enfermedades.

En resumen, se secuencian los exones, y aquellos que son más grandes y no se pueden secuenciar de una vez, se secuencian por segmentos y posteriormente se realiza el ensamblado. Se debe tener en cuenta que si se ha secuenciado más de un exón, los segmentos una vez ensamblados se agrupan en regiones disjuntas, es decir, una por exón.

A la hora de secuenciar un exón, si este es muy largo (aproximadamente más de 600 nucleótidos) se divide en segmentos. Cada uno de estos segmentos se secuencia, como mínimo, dos veces. Así se obtienen dos secuenciaciones diferentes de la secuencia que se quiere obtener. Esto permite tener redundancia para la detección de fallos puntuales de secuenciación.

En el marco de un análisis de variaciones para el soporte al diagnóstico, se selecciona un gen y los exones a secuenciar. Para la localización de los exones se utiliza una unidad de transcripción, que es una lista de posiciones que indican que regiones del gen son exones. Una vez secuenciados todos los segmentos necesarios se procede a reconstruir la secuencia, que consiste en ensamblar los segmentos y obtener la secuencia resumen, o representativa, de todos ellos. A la secuencia resumen obtenida se le denomina consenso.

4.2.2. Limpieza de secuencias

Debido a la introducción de errores en los extremos de cada uno de los segmentos secuenciados, es necesario realizar una limpieza de los electroferogramas obtenidos del proceso de secuenciación.

Cuando todos los segmentos se secuencian con la misma máquina secuenciadora, es posible llevar a cabo la limpieza de una manera eficiente mediante la aplicación de una misma configuración de limpieza. Para ello se deben establecer un conjunto de parámetros que eliminen aquellos conjuntos de señales que no sean legibles y, por lo tanto, no se pueden traducir a nucleótidos de una manera fiable.

La realización de la limpieza de los segmentos de manera simultánea reduce el procesamiento manual evitando tener que repetir el proceso para cada uno de los segmentos secuenciados.

4.2.3. Revisión de secuencias

Debido a la introducción de errores puntuales en los electroferogramas obtenidos por las máquinas secuenciadoras, es necesario realizar una revisión de la secuencia de manera manual. Para ello, los biólogos observan los electroferogramas y, mediante su experiencia en el dominio, son capaces de distinguir el valor correcto.

Con el fin de solventar algunos de estos errores de manera eficiente, el proceso de secuenciación de un segmento se realiza varias veces, ofreciendo redundancia, con el fin de realizar la corrección de errores manera directa. Cuando un segmento se ha secuenciado varias veces, al realizar el ensamblaje y calcular el consenso, muchas de ellas se corrigen automáticamente.

Una vez realizadas estas correcciones automáticas, los biólogos revisan el consenso obtenido y visualizan simultáneamente los electroferogramas secuenciados. Al revisar únicamente el consenso final se evitan la revisión de todos los segmentos secuenciados.

4.3. Especificación de requisitos

En la fase de Tratamiento se plantean tres casos de usos diferentes para la obtención de la secuencia del paciente. Por un lado se soporta la introducción de la secuencia de nucleótidos completa. Este caso de uso se puede utilizar en el caso de que se resolvieran todos los problemas de las máquinas secuenciadoras, o en el caso en el que se ha secuenciado el gen completo y se ha realizado un preproceso de los segmentos obtenidos. Por otro lado, se soporta la introducción de la secuencia de nucleótidos por exones. Este caso de uso se puede utilizar en el caso en el que se ha secuenciado diversos exones y ya se ha realizado un preproceso de los datos. Por último, se soporta la introducción de la información obtenida directamente de las máquinas secuenciadoras sin ningún preprocesamiento.

En la fase Tratamiento se llevan a cabo todas las tareas incluidas en el caso de uso “Obtener Secuencia del Paciente”, que esta incluido en el caso de uso “Obtener secuencias” (Figura 4.3).

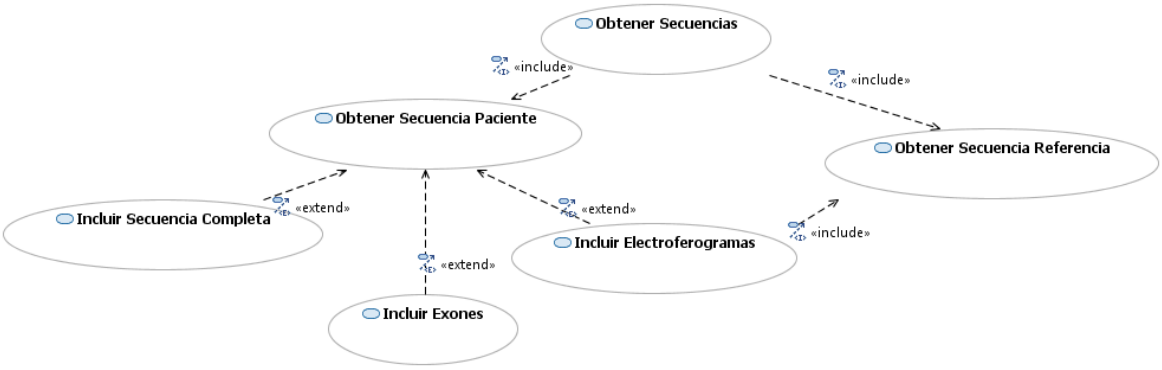


Figura 4.3: Caso de Uso "Obtener Secuencias"

Caso de Uso "Obtener Secuencia del Paciente"

El propósito del caso de uso es obtener una secuencia genética que represente el gen seleccionado del paciente que se quiere analizar. Para ello se introducen sus datos genéticos y, según la naturaleza de estos, se llevan a cabo las tareas necesarias para recomponer la secuencia genética objetivo de análisis.

- Actores que intervienen: El biólogo interviene en el caso de uso para proporcionar la entrada de datos genéticos del paciente y, cuando los datos genéticos lo requieren, también para revisar secuencias de nucleótidos. La HGDB proporciona información sobre la secuencia de referencia.
- Precondiciones: El gen que se está analizando ha sido seleccionado por el biólogo y por lo tanto es conocido en el sistema.
- Postcondiciones: Se obtiene la secuencia del gen del paciente que se quiere analizar y es correcta porqué está limpia, revisada y ensamblada en su correcta posición.
- Casos de uso que lo extienden: Incluir Secuencia Completa, Incluir Exones e Incluir Electroferogramas.
- Casos de uso que le incluye: Obtener Secuencias

En caso de uso está formado por los siguientes pasos:

1. El biólogo indica al sistema el gen que se ha seleccionado para el análisis.
2. El biólogo selecciona como va a introducir los datos genéticos del paciente. Existen tres opciones: 1) Incluir una secuencia completa de nucleótidos; 2) Incluir el conjunto de exones, es decir, de subsecuencias de nucleótidos que forman el gen; y 3) Incluir los segmentos obtenidos de las máquinas secuenciadoras.

3. Según la forma en la que se han introducido los datos genéticos en el paso anterior se ejecutan un conjunto de acciones u otras. Para ello se selecciona el caso de uso correspondiente.
4. El sistema devuelve al usuario la secuencia del paciente correcta y completa.

Caso de Uso "Incluir Secuencia Completa"

El propósito del caso de uso es obtener la secuencia genética del paciente a partir de la introducción de una secuencia de caracteres.

- Actores que intervienen: El biólogo interviene en el caso de uso para proporcionar la entrada de la secuencia
- Precondiciones: La entrada debe ser una cadena de caracteres o la ruta de un fichero.
- Postcondiciones: Se proporciona una secuencia genética completa y correcta a partir de la entrada introducida.
- Caso de uso al que extiende: Obtener Secuencia Paciente

El caso de uso está formado por los siguientes pasos:

1. El biólogo introduce los datos necesarios para introducir la secuencia.
2. El sistema lee una secuencia de caracteres introducida directamente o lee el fichero indicado.
3. El sistema comprueba que la secuencia no contiene ningún caracter no permitido.
4. El sistema devuelve dicha secuencia de caracteres como la secuencia genética de nucleótidos correcta y completa.

Caso de Uso "Incluir Exones"

El propósito del caso de uso es obtener la secuencia genética del paciente a partir de la introducción de una o varias secuencias de caracteres. Para ello, es necesario leer los caracteres introducidos y según su estructura diferenciar los exones que forman la secuencia.

- Actores que intervienen: El biólogo interviene en el caso de uso para proporcionar la entrada de exones.
- Precondiciones: El formato de entrada debe ser conocido por el sistema.

- Postcondiciones: Se proporciona una secuencia genética completa y correcta a partir de la entrada introducida.
- Caso de uso al que extiende: Obtener Secuencia Paciente

El caso de uso está formado por los siguientes pasos:

1. El biólogo introduce los datos necesarios para introducir los exones.
2. El sistema lee la entrada introducida e identifica el formato.
3. El sistema identifica los exones del gen a partir de los datos de entrada.
4. El sistema comprueba que la secuencia no contiene ningún caracter no permitido.
5. El sistema devuelve la secuencia genética de nucleótidos correcta y completa.

Caso de Uso "Incluir Electroferogramas"

El propósito del caso de uso es obtener la secuencia genética del paciente a partir de un conjunto de ficheros que contengan los electroferogramas, y los nucleótidos asociados, de cada segmento de secuenciación. Para ello, es necesario leer los archivos, y realizar las operaciones necesarias para identificar las partes secuenciadas del gen.

- Actores que intervienen: El biólogo interviene en el caso de uso para proporcionar la entrada de electroferogramas así como para revisar la secuencia de nucleótidos. La HGDB proporciona la secuencia de referencia del gen seleccionado.
- Precondiciones: El gen que se está analizando ha sido seleccionado por el biólogo y, por lo tanto, es conocido en el sistema.
- Postcondiciones: Se proporciona una secuencia genética completa y correcta a partir de la entrada introducida.
- Caso de uso al que extiende: Obtener Secuencia Paciente.

El caso de uso está formado por los siguientes pasos:

1. El biólogo introduce los datos necesarios para introducir los segmentos secuenciados.
2. El sistema obtiene la secuencia de referencia del gen seleccionado.
3. El sistema ensambla los segmentos de nucleótidos obtenidos contra la secuencia de referencia.
4. El sistema calcula la secuencia consenso de todos los segmentos ensamblados.
5. El biólogo revisa la secuencia consenso obtenida por el sistema.
6. El sistema devuelve la secuencia genética de nucleótidos correcta y completa.

Caso de Uso "Obtener Secuencia de Referencia"

El propósito del caso de uso es obtener una secuencia que represente al gen genérico que se quiere analizar. Para ello se obtiene el conocimiento del sistema de información genómico.

- Actores que intervienen: La HGDB proporciona información sobre la secuencia de referencia.
- Precondiciones: El gen que se está analizando ha sido seleccionado por el biólogo y, por lo tanto, es conocido en el sistema.
- Postcondiciones: Se obtiene la secuencia de referencia del gen.
- Casos de uso que lo incluyen: Obtener Secuencias e Incluir Electroferogramas.

En caso de uso está formado por los siguientes pasos:

1. El sistema el realiza una petición a la HGDB indicando el gen seleccionado para el análisis.
2. La HGDB devuelve la secuencia de referencia del gen seleccionado.
3. El sistema devuelve la secuencia de referencia.

4.4. Modelado Conceptual

La fase de Tratamiento es la responsable de reconstruir una secuencia de nucleótidos única a partir de los electroferogramas obtenidos por las máquinas secuenciadoras. Nuestro objetivo es leer los ficheros obtenidos por dichas máquinas y obtener la secuencia genética del gen del paciente objetivo de análisis.

En la Figura 4.4 se muestra una vista del CSHG que contiene los conceptos biológicos involucrados en la ejecución de esta fase. En este modelo, la entidad *Gene*, modela el concepto genérico de gen como región de un cromosoma. Los atributos que describen todos los genes son: *id_symbol*, que es un identificador alfanumérico de HGNC [32]; id_*HUGO*, que es el código universal del gen también de HGNC; *official_name*, que es el nombre oficial del gen; *summary*, que es una breve descripción del gen; *chromosome*, que es el número del cromosoma al que pertenece el gen; y *locus*, que es la región del cromosoma donde se encuentra el gen.

Las diferentes instancias de un gen se representan por la entidad conceptual *Allelic*. Los atributos que describen todas las instancias de tipo Allelic son: *ord_num*, que es un identificador para diferenciar las diferentes instancias; *start_position*, que es la posición inicial del gen respecto al cromosoma; *end_position*, que es la posición final del gen respecto al cromosoma; *strand*, que indica cual de las dos hebras se ha utilizado para expresar la secuencia genética.

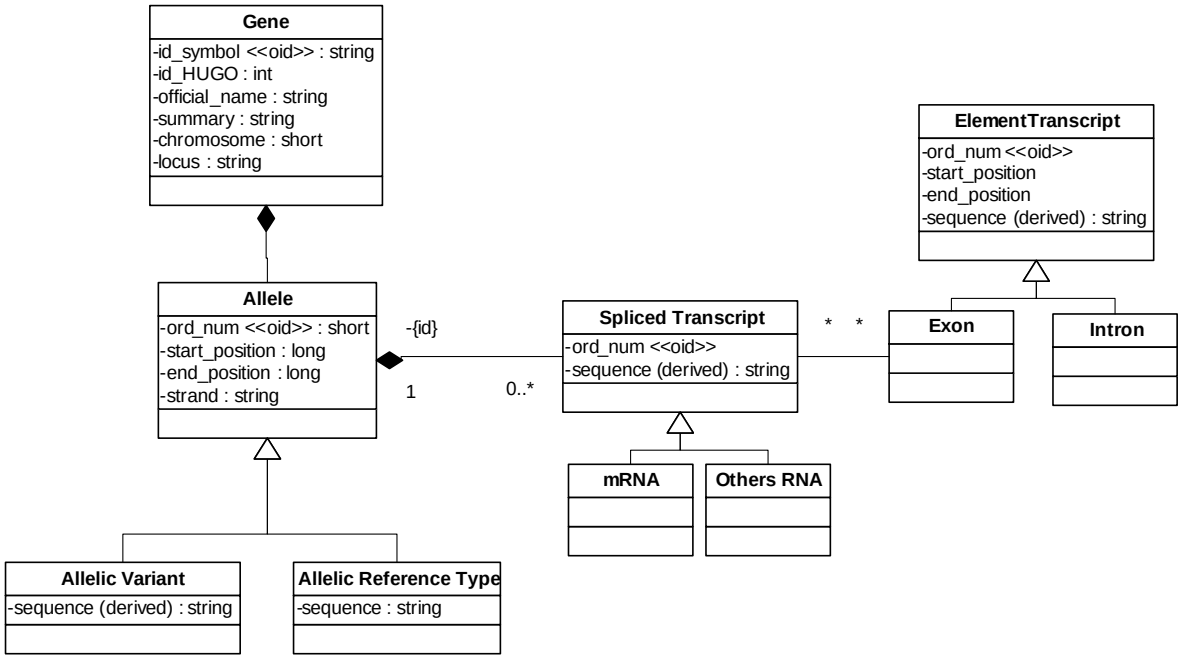


Figura 4.4: Esquema Conceptual del Genoma Humano (CSHG)

La entidad *Allelic* tiene dos especializaciones: *Allelic Variant* y *Allelic Reference Type*. *Allelic Variant* representa la información del gen de un individuo. Sin embargo, cabe destacar que esta información, aunque representa la información genética de uno o varios individuos, no suele ir asociada a ellos en el modelo, sino que se utiliza para describir algunas propiedades que se han observado específicamente en sus genotipos. *Allelic Reference Type* representa la secuencia de referencia del gen, que a pesar de tratarse de una secuencia simbólica, también es una instancia específica de un *Gene* y por lo tanto es de tipo *Allelic*. Ambas entidades tienen el atributo *sequence*, que contiene los nucleótidos de la secuencia genética.

La clase *SplicedTranscript* modela el concepto de transcrito, es decir, como se divide un gen y como se organiza para el proceso de transcripción para la producción de proteínas. Esta entidad se especializa en dos entidades: mRNA, que representa al ARN mensajero que sale de la célula y codifica una proteína, o Others RNA, que modela otras funcionalidades que puede tener el ARN.

La entidad conceptual *ElementTranscript*, modela los diferentes segmentos en los que se divide la secuencia de ADN de un gen. Estos segmentos se describen mediante los atributos: *ord_num*, que es un identificador para diferenciar un segmento de otro; *start_position* y *end_position*, que indican la posición inicia y final del segmento respecto al gen respectivamente; y *sequence*, que representa la secuencia genética del segmento. Los segmentos *ElementTranscript* son de dos tipos: de tipo *Exon*, representando aquellos segmentos que realmente se transcriben, y de tipo *Intron*, representando aquellos segmentos que no se transcriben. Por esta razón, la entidad *SplicedTranscript* está únicamente relacionada con la entidad *Exon*, pero no con la entidad *Intron*. Esto significa que cada *SplicedTranscript* está asociado a un conjunto de exones que unidos forman el transcrito del gen, es decir, el atributo *sequence* es la concatenación de todas

las secuencias genéticas, también el atributo *sequence*, de dichos exones.

El modelo conceptual de esta fase está basado en los conceptos biológicos plasmados en esta vista. A partir de este modelo conceptual, el objetivo es reutilizar aquellos conceptos que sirvan para llevar a cabo las tareas necesarias para completar la fase de Tratamiento. Para ello, se utilizan los mismos conceptos biológicos pero simplificados y adaptados a las tareas que se quieren llevar a cabo.

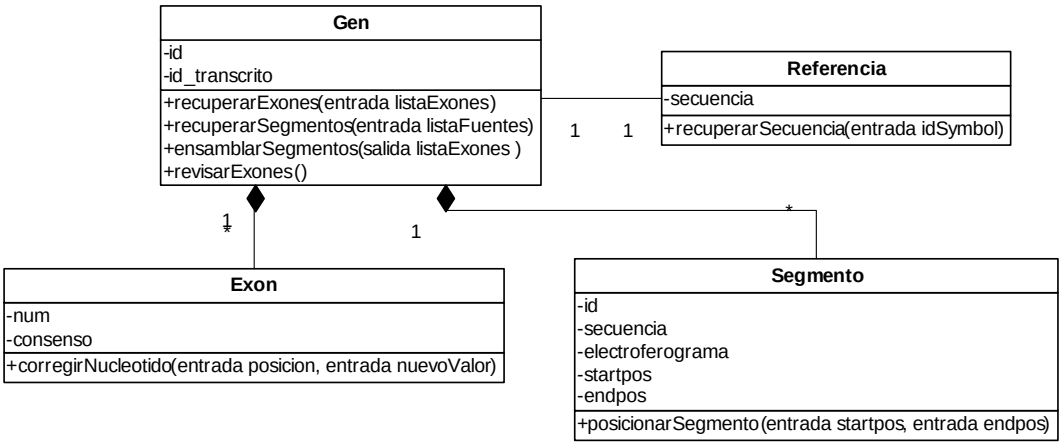


Figura 4.5: Modelo Conceptual Tratamiento

El modelo conceptual diseñado está formado por cuatro entidades conceptuales (Figura 4.5). En primer lugar, la entidad *Gen,* que representa el gen del paciente cuyo genotipo se quiere secuenciar. En relación al CSHG, es la entidad *Allelic Variant* la entidad que modela una instancia de un gen genérico que pertenece a un individuo. En el modelo de la fase Tratamiento, debido a que solo hay una única instancia de un individuo, se pierde la especialización y se denomina *Gen*e. La entidad *Gene* está formada por dos atributos, *id* y *genotipo*. El atributo *id* representa el identificador del gen que se utiliza en toda la comunidad científica, lo que en el CSHG es el id_*symbol* de la entidad *Gene*. El atributo *genotipo*, es la secuencia obtenida en el proceso de secuenciación, es decir, los nucleótidos que forman la secuencia del gen del paciente, lo que en CSHG es *sequence* en la entidad *AllelicVariant.*

Debido a las limitaciones tecnológicas, el gen del paciente se secuencia por segmentos. Un *Gene* está compuesto por todos aquellos *Segmentos* obtenidos de la secuenciación. Cada *Segmento* tiene un atributo *id*, que es el identificador que se utiliza para diferenciarlo del resto de *Segmentos,* y un atributo *secuencia*, que es la sucesión de nucleótidos del *Segmento*. Debido a que esta consideración se debe hacer por razones tecnológicas, no se tiene ningún concepto asociado al CSHG.

Para poder reconstruir el *Gen* del paciente a partir de los *Segmentos* secuenciados, es necesario utilizar una *Referencia* para localizar la posición del segmento, ya que, de lo contrario, no es posible ordenar ni unir los segmentos. Una *Referencia* tiene un atributo *id*, que representa el identificador de la secuencia simbólica que se utiliza en toda la comunidad científica, y un atributo *secuencia*, que contiene la secuencia completa de nucleótidos que se supone representativa de la secuencia del *Gene* de todos los individuos de la misma especie, en este caso, la

especie humana. En relación al CSHG, la entidad *Referencia* representa el mismo concepto que la clase *Allelic Reference Type*. Del mismo modo que ha ocurrido con la entidad *Gene*, en la fase Tratamiento solo tendremos una instancia de la clase *Referencia* para el *Gen*.

Debido a las limitaciones tecnológicas, la secuenciación de un gen es costosa, por eso, no se realiza de manera completa y se seleccionan únicamente las partes del gen que son necesarias para el análisis. El biólogo selecciona un transcrito del gen que quiere analizar, identificado por el atributo *id_transcrito*. Este transcrito indica qué partes del gen son codificantes para la proteína (exones), y por tanto es necesario secuenciar, y cuales se pueden descartar del análisis. En el CSHG este identificador del transcrito se encuentra en el atributo *id* de la relación entre *Allelic* y *SplicedTranscript*.

Una vez seleccionado el transcrito y realizada la secuenciación basada en él, los *Segmentos* obtenidos se ensamblan, utilizando la referencia. Este ensamblaje se lleva a cabo para posicionar cada segmento en su lugar y poder reconstruir cada uno de los *Exones* de manera ordenada dentro del *Gen*. El atributo *num* contiene el numeral de esta ordenación, y el atributo *consenso* contiene la secuencia genética consenso a partir de todos los segmentos secuenciados del *Exon*.

Por lo tanto, los *Segmentos* en los que se divide el *Gen* que se secuencian, en realidad son *Segmentos* que representan los *Exones*. La *Referencia* del gen se utiliza únicamente para reconstruir cada uno de los *Exones*.

Para llevar a cabo la fase de Tratamiento y proporcionar la funcionalidad expresada en los casos de uso, es necesario proveer a las clases de métodos que ejecuten la funcionalidad necesaria (Figura 4.5).

Para llevar a cabo el caso de uso "Obtener Secuencia Paciente" es necesario crear una instancia de la clase Gen y según la selección de caso de uso por el biólogo se pueden ejecutar los métodos:

Caso de uso: "Incluir Secuencia Completa" Para llevar a cabo este caso de uso la clase se utiliza el constructor de la clase Gen:

1. Crear un nuevo Gen.
2. Inicializar la secuencia del gen.

Caso de uso: "Incluir Secuencia Exones" Para llevar a cabo este caso de uso se reutiliza el método de la clase Gen Recuperar Exones:

recuperarExones(entrada listaExones)

Este método se encarga de crear tantos exones como componentes tenga la lista de Exones introducida por el argumento de entrada. Para este caso de uso, la lista contendrá tantos elementos como exones se hayan secuenciado. Para cada elemento de la lista el método se encarga de:

1. Crear un nuevo Exon.
2. Incluir el ordinal num de acuerdo con la posición del elemento en la lista.
3. Inicializar la secuencia consenso del Exon de acuerdo con el contenido del elemento en la lista.

Caso de uso: "Incluir Segmentos" Para llevar a cabo este caso de uso se necesitan diversos métodos de la clase Gen, la clase Referencia, la clase Segmento y la clase Exon. Estos métodos se encargan de llevar a cabo cada una de las acciones requeridas para reconstruir la secuencia genética del paciente a partir de los segmentos de nucleótidos introducidos. Estos métodos son los siguientes:

1. Métodos de la clase Gen: Recuperar Segmentos, Ensamblar Segmentos y Revisar Exones:

   recuperarSegmentos(entrada listaFuenteSegmentos)

   Este método se encarga de crear tantos segmentos como componentes tenga la lista de fuentes de Segmentos introducida por el argumento de entrada. La lista contiene la ruta de los ficheros que contienen los segmentos. Para cada elemento de la lista el método se encarga:

   a) Crear un nuevo Segmento.
   b) Leer el fichero cuya ruta se encuentra en el contenido del elemento de la lista.
   c) Inicializar el electroferograma y la secuencia del segmento.

   ensamblarSegmentos(salida listaExones)

   Este método se encarga de crear de identificar cada segmento del Gen en su correcta posición en la secuencia de Referencia, obtener el consenso de todos los segmentos y devolver la lista de los consensos obtenidos (que representan los exones). El método se encarga de:

   a) Alinear cada uno de los segmentos con la secuencia de referencia y posicionarlo respecto a la ella.
   b) Calcular el consenso de todos los segmentos.
   c) Crear una lista con todos los consensos disjuntos obtenidos.

   revisarExones()

   Este método se encarga de mostrar al usuario el consenso de los exones para que pueda corregir los posibles errores. El método se encarga de:

   a) Mostrar valor del consenso y los electroferogramas asociados.
   b) Cambiar los valores de los nucleótidos erróneos.

2. Métodos de la clase Referencia: Recuperar Secuencia

   recuperarSecuencia(entrada idSymbol)

   Este método recupera la secuencia de referencia completa del gen indicado. El método se encarga de:

   a) Realizar una consulta a la base de datos indicando el identificador del gen.

   b) Inicializar la secuencia a partir de los datos obtenidos.

3. Métodos de la clase Segmento: Limpiar Segmento, Posicionar Segmento

   limpiarSegmento(entrada parametrosLimpieza)

   Este método se encarga de limpiar los extremos del segmento. El método se encarga de:

   a) Eliminar los nucleótidos necesarios del inicio y final del segmento de acuerdo con los parámetros de limpieza.

   posicionarSegmento(entrada startpos, entrada endpos)

   Este método se encarga de posicionar el segmento en su correcta posición respecto a la secuencia de referencia. El método se encarga de:

   a) Inicializar las posiciones inicial y final del segmento.

4. Métodos de la clase Exon: Corregir Nucleótido

   corregirNucleotido(entrada posición, entrada nuevoValor)

   Este método se encarga de corregir el valor del nucleótido que el biólogo considera incorrecto. El método se encarga de:

   a) Inicializar el valor del carácter del consenso de la posición indicada con el nuevo Valor.

4.5. Implementación

Para llevar a cabo la implementación de esta fase se ha integrado una herramienta de edición, limpieza y ensamblaje de secuencias llamada Sequencher. Se ha seleccionado esta herramienta por ser la herramienta principal de estudio y la herramienta que utilizan en el Instituto de Medicina Genómica IMeGen. La herramienta Sequencher puede utilizarse para llevar a cabo la reconstrucción de la secuencia genética de esta fase. Para ello, se seleccionan un conjunto de tareas que ofrece esta herramienta: ensamblaje de muestras, limpieza de muestras y visualización gráfica de electroferogramas para la corrección del consenso.

La información obtenida por las máquinas secuenciadoras se procesa mediante la herramienta Sequencher, y se implementa un traductor para transformar las salidas de Sequencher en términos del modelo conceptual (Figura 4.6).

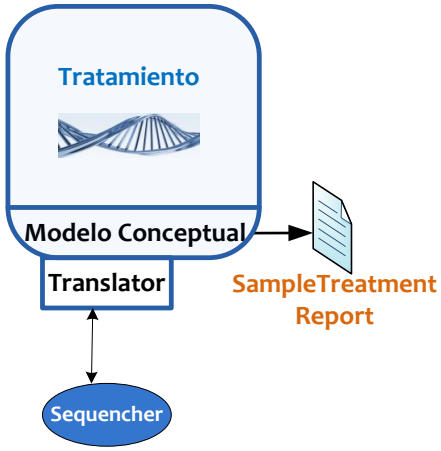


Figura 4.6: Fase Tratamiento

4.5.1. Modelo Conceptual Reporte de Resultados

Se ha diseñado un modelo conceptual de Reporte de Resultados, *SampleTreatmentReport*, para expresar los resultados de la fase de Tratamiento de manera precisa y para guiar el proceso de flujo de datos.

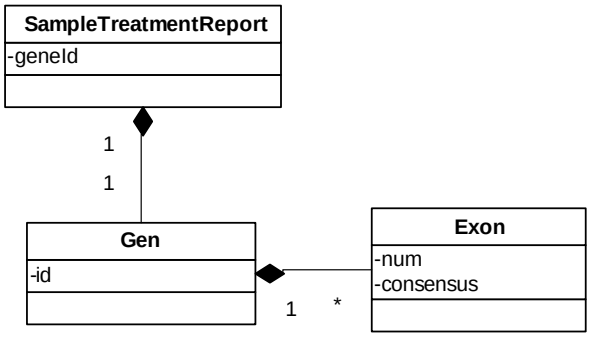


Figura 4.7: Modelo Conceptual SampleTreatmentReport

El modelo conceptual de la Figura 4.7 muestra la información de una secuencia genética reconstruida. Este modelo conceptual es un vista del modelo conceptual de la fase de Tratamiento. La entidad conceptual *SampleTreatmentReport* representa el reporte generado después de realizar el tratamiento o reconstrucción de la muestra de ADN del paciente, y el atributo *geneId* contiene el identificador del gen de dicho tratamiento. La entidad conceptual *Gen* representa el gen del paciente que se quiere analizar y el atributo *id* representa el identificador del gen. Un *SampleTreatmentReport* debe tener uno y solo un elemento *Gen*, ya que la secuencia reconstruida pertenece a un único gen. La entidad conceptual *Exon* representa los diferentes trozos que tienen significado en el proceso de transcripción y que por lo tanto se han secuenciado para realizar el análisis. Esta entidad tiene dos atributos: *num*, que representa el ordinal del exón respecto del resto, y *sequence*, que representa la secuencia de nucleótidos del exón. Un *Gen* puede tener múltiples elementos *Exon*, pero un *Exon* solo pertenece a un *Gen*.

En la Figura 4.8 se muestra como se integra la herramienta Sequencher en el framework mediante la definición del elemento de traducción. En este caso solo es necesario traducir la salida del Sequencher en términos del modelo conceptual, por lo que únicamente se diseña el módulo Output. El SequencherOutputTranslator se encarga de establecer la relación entre la salida del Sequencher y el modelo conceptual SampleTreatmentReport.

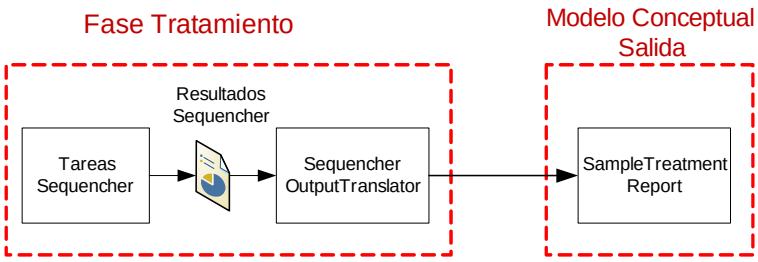


Figura 4.8: Integración de Sequencher

4.5.2. Implementación del Modelo Conceptual Reporte de Resultados

La implementación de la fase de Tratamiento se ha desarrollado en Java y el modelo conceptual diseñado se ha implementado de dos formas:

- Implementación mediante clases Java: Esta implementación permite el intercambio de información en el contexto del framework Diagen. En este caso, el flujo de información se consigue mediante la instanciación de los objetos del modelo conceptual.
- Implementación mediante XMLSchema: Esta implementación permite la introducción de datos y la obtención de resultados mediante el formato XML. En este caso, se ha utilizado la tecnología JAXB (Java Architecture for XML bindings) [22] de manera que se instanciando los objetos del modelo conceptual en Java se pueda crear el archivo XML asociado. Los pasos para utilizar JAXB son los siguientes:

  1. Descripción del modelo conceptual mediante XMLSchema: Se declaran todos los objetos, sus propiedades y los tipos y restricciones de éstos.
  2. Obtener objetos Java a partir del XMLSchema: Mediante la utilidad de JAXB, xjc, se compila el XMLSchema que da lugar al conjunto de objetos Java.
  3. Definición operaciones de transformación: Se implementan las operaciones de marshalling, que transforman los objetos Java a un archivo XML y las operaciones de unmarshalling, para transformar un archivo XML a objetos Java.

Una vez implementada la funcionalidad y los artefactos necesarios para expresar los resultados en términos del modelo conceptual, la fase puede ejecutarse de manera independiente. En la figura 4.9, se puede observar un ejemplo de la salida XML que proporciona la ejecución de la fase de Tratamiento de manera independiente. Para ello se ha utilizado una muestra del gen BRCA1, para la cual se han secuenciado los 22 exones del gen.

```
<SampleTreatmentReport idGene="BRCA1">
    <Query id="BRCA1">
        <Exon num="0">
            <consensus>GCACTTTATGGCAAACT...AGCTGCTGAGATGGGTATTCTTTGA</consensus>
        </Exon>
        <Exon num="1">
            <consensus>GACGTTGTCATTAGTTA...CTCTACTAACAGAATTGACCTTACA</consensus>
        </Exon>
        ...
        <Exon num="22">
            <consensus>CCCTGGAGTCGATTGAT...AATTTATTCAGGCTGTTGTTGGCTT</consensus>
        </Exon>
    </Query>
</SampleTreatmentReport>
```

Figura 4.9: Ejemplo XML SampleTreatmentReport

Capítulo 5

Fase Alineamiento

La fase Alineamiento es la fase que compara la secuencia genética del paciente con la secuencia de referencia, con el fin de obtener la lista de diferencias entre ellas. Para la detección de diferencias es necesario llevar a cabo una alineación de ambas secuencias genéticas, es decir, alinear la cadena de texto de la secuencia objetivo con la cadena de texto de la secuencia de referencia. Esta alineación consiste en identificar la posición de la referencia donde empieza la secuencia objetivo y averiguar cómo difiere la secuencia objetivo de la secuencia de referencia.

El alineamiento entre secuencias genéticas se realiza mediante algoritmos de alineamiento genético. Estos algoritmos, además de proporcionar la posición de la referencia sobre la que se alinea el objetivo, también se encargan de introducir desplazamientos en ambas secuencias de manera que las cadenas de texto coincidan en mayor medida. De esta forma, revisando el alineamiento obtenido por un algoritmo es posible obtener el conjunto de diferencias entre la secuencia genética del individuo y la secuencia de referencia.

En este capítulo, en primer lugar, se proporciona un conjunto de conceptos biólogicos para comprender el proceso de identificación de diferencias en una secuencia de ADN. Los algoritmos de alineamiento genético incorporan nuevas concepciones a los algoritmos de alineamiento textuales, ya que, se tienen en cuenta las propiedades del ADN presentes en las secuencias genéticas.

En segundo lugar, se identifican los problemas encontrados a la hora de detectar las diferencias entre las secuencias genéticas a comparar y las soluciones que se adoptan para solventarlas. En el diseño de esta fase se abordan las limitaciones introducidas por los algoritmos de alineamiento: 1) Coste computacional de comparación entre secuencias; y 2) Indeterminismo a la hora de expresar una diferencia localizada. En esta fase, también se abordan las limitaciones introducidas por las máquinas secuenciadoras: 3) El problema de expresividad que tienen los electroferogramas al no ofrecer de manera precisa toda la información genética del individuo.

En tercer lugar, se plantean los diferentes casos de uso necesarios para la obtención de la lista de diferencias. Se presentan dos aproximaciones diferentes: 1) la detección de diferencias

mediante la comparación entre secuencias; y 2) La búsqueda dirigida de una lista de diferencias ya localizadas.

En cuarto lugar, se explica el modelo conceptual diseñado y su correspondencia con el CSHG. Se presentan las entidades conceptuales y los métodos necesarios para llevar a cabo todos los casos de uso para la detección de diferencias en una secuencia genética.

Por último, se explica la implementación llevada a cabo y el modelo conceptual de Reporte de Resultados AlignmentReport, utilizado para expresar los resultados de la fase de Alineamiento de manera precisa.

5.1. Background biológico

La secuencia genética y la secuencia de Referencia

Cada ser humano tiene 23 pares de cromosomas, donde cada cromosoma proviene de un progenitor, es decir, para cada par de cromosomas uno es heredado del padre y otro de la madre.

Por ese motivo, la secuencia textual del ADN de un individuo representa a ambos cromosomas, es decir, cada caracter de la secuencia representa el valor de dos nucleótidos. Una secuencia textual representa al mismo tiempo dos secuencias, una que representa al cromosoma del padre y otra al cromosoma de la madre. En aquellas posiciones en las que el cromosoma del padre contiene el mismo nucleótido que el cromosoma de la madre, la representación en la secuencia viene dado por el valor del nucleótido común: A, C, T o G. Sin embargo, cuando el nucleótido difiere en cada cromosoma, la representación en la secuencia debe representar ambos valores. Para representar las diferentes combinaciones entre los nucleótidos A, C, T y G, existen unos códigos predeterminados por la IUPAC (International Union of Pure and Applied Chemistry) [16], llamados códigos IUB. En la Tabla 5.1 pueden verse algunos de los códigos determinados por la UIPAC, en este caso, los que nos interesan son los códigos que representan uno nucleótido y los que representan dos nucleótidos simultáneamente.

Código	Definición	Código	Definición
A	Adenina	M	AC
C	Citosina	S	GC
T	Timina	W	AT
G	Guanina	V	GAC
R	AG	B	GTC
Y	CT	H	ATC
K	GT	D	GAT

Cuadro 5.1: Códigos IUB

Los algoritmos de alineamiento genéticos

Los algoritmos de alineamiento son algoritmos de análisis de caracteres que tienen en cuenta las cadenas de texto a la hora de realizar el alineamiento. Concretamente, al tratarse de algoritmos de alineamiento genéticos además se tienen en cuenta algunas propiedades de las secuencias del ADN. Estas propiedades en ocasiones son necesarias para poder detectar ciertos casos que con los algoritmos de alineamiento convencionales no es posible reconocer. Algunas de estas propiedades son:

1. Las secuencias tienen un conjunto de caracteres definidos: Únicamente están permitidos los códigos IUB (caracteres A, C, T, G y los caracteres V, B, H, D, K, S, W, M, Y, R).
2. Existen caracteres (K, S, W, M, Y, R) que representan el valor de dos nucleótidos simultáneamente, por lo que al realizar la comparación, se debe tener en cuenta que si el caracter difiere del de la secuencia de referencia, existen diferentes posibilidades: 1) Ningún nucleótido coincide; 2) Un nucleótido coincide y el otro no; o 3) El valor es ambiguo en ese punto. En este tipo de casos un algoritmo de alineamiento convencional simplemente detecta una diferencia, pero no profundiza en su naturaleza.

Las variaciones genéticas

El concepto de variación genética, se refiere a la diferencia que existe entre la secuencia de un individuo y la secuencia de referencia, que se supone es una secuencia representativa del ADN de todos los individuos.

Este tipo de cambios, pueden deberse a dos causas: 1) La variación se hereda de un progenitor; y 2) Se produce un error en el proceso de herencia y se introducen cambios en la copia del cromosoma.

Las variaciones se pueden ser de tres tipos:

- Inserciones: Se ha añadido uno o varios nucleótidos en el ADN.
- Deleciones: Se ha eliminado uno o varios nucleótidos en el ADN.
- Indels: Se ha producido un cambio en uno o varios nucleótidos del ADN.

Dependiendo del cromosoma en el que se produzca el cambio una variación se clasifica en:

- Variaciones en homocigosis: Se produce la misma variación en el cromosoma del padre y en el cromosoma de la madre.
- Variaciones en heterocigosis: Se produce una variación únicamente en el cromosoma del padre o en el de la madre.

5.2. Problemas Detectados y Soluciones

A fin de proporcionar una lista de diferencias que representen todos los cambios que existen en una secuencia genética es necesario resolver los problemas computacionales, los problemas de indeterminismo y los problemas de las máquinas secuenciadoras. Para ello se proponen las siguientes soluciones para cada uno de los problemas:

- Problemas computacionales: Los algoritmos de alineamiento pretenden realizar búsquedas de secuencias contra bases de datos de tamaño considerable, por lo que el coste computacional y temporal se hace inviable. La solución que se propone es acotar la búsqueda de manera que se realice un alineamiento dirigido menos costoso computacionalmente.
- Problemas de indeterminismo: Los algoritmos de alineamiento ofrecen diferentes conclusiones acerca de las variaciones localizadas. La solución que se propone es realizar una búsqueda dirigida mediante la caracterización de las variaciones mediante regiones adyacentes.
- Superposición de cromosomas en los electroferogramas: Se propone como solución la aproximación de la inferencia haplotípica basada en la secuencia de referencia. Esta aproximación consiste en intentar derivar, a partir del electroferograma y la ocurrencia de una variación, cada uno de los cromosomas, teniendo como base la secuencia de referencia.

5.2.1. Búsqueda acotada de variaciones: Reducción del problema

Los algoritmos de alineamiento genético actuales localizan una secuencia de nucleótidos en genomas completos de diferentes los organismos secuenciados (ser humano, mosca dorsophila, etc.). Sin embargo, estos algoritmos, son muy costosos computacionalmente hablando, ya que tardan mucho y consumen mucha memoria.

Algunas aproximaciones como el algoritmo BLAST [2], consigue reducir el coste computacional utilizando búsquedas heurísticas. Sin embargo, esta aproximación, aunque es útil para la ejecución de algunas tareas bioinformáticas, los resultados que ofrecen no son totalmente precisos.

El carácter del análisis que se trata en esta tesis de máster requiere de resultados precisos, ya que se trata de localizar las variaciones de un individuo y ofrecer un diagnostico, hecho que no da cabido a la ambigüedad. El objetivo es reducir el coste computacional pero no a costa de obtener resultados precisos.

La solución que se propone para reducir este coste y no perder precisión en los resultados es utilizar la naturaleza de los análisis y realizar solo las búsquedas estrictamente necesarias. El análisis genético que se lleva a cabo en esta fase es una análisis de una o varias secuencias contra una secuencia de referencia. En otras palabras, se conoce la naturaleza de las consultas

(uno o varios exones), y se conoce la secuencia contra la que se deben realizar el alineamiento (secuencia de referencia del gen seleccionado).

Por este motivo, se ha acotado la búsqueda de varias secuencias contra varias secuencias a una o varias secuencias pequeñas a una única secuencia. La búsqueda consiste en localizar secuencias aproximadamente entre 200 y 5000 nucleotidos contra una secuencia que tiene como máximo 31000 nucleótidos. Además, las secuencias a alinear siempre deben tener éxito y solo deben tener una solución posible, ya que cada exón secuenciado pertenece a una región de la secuencia de referencia.

Por otro lado, se acota la búsqueda ya que el alineamiento se realiza entre secuencias con una variabilidad muy baja. En este tipo de análisis se tiene en cuenta que un ser humano tiene una sustitución de un nucleótido cada 200 nucleótidos en comparación con la secuencia de referencia.

5.2.2. Búsqueda dirigida variaciones: Búsqueda por regiones adyacentes

La ambigüedad en la caracterización de variaciones es un problema presente en los diferentes repositorios genómicos actuales. Cuando un investigador biólogo identifica una variación y mediante sus experimentos concluye que está asociada a un fenotipo concreto, este indica la posición en la que ha localizado la variación, el valor original que tiene la referencia y el valor que se ha detectado que ha cambiado.

El problema reside en el hecho de que esta caracterización: el tipo, la posición y el valor de la variación dependen del algoritmo de alineamiento utilizado para comparar con la referencia. Como consecuencia, puede ocurrir que un biólogo identifique una variación (por ejemplo una sustitución de dos nucleótidos) y otro identifique dos variaciones (por ejemplo una deleción de un nucleótido en una posición y una inserción de un nucleótido en otra posición). Cuando en realidad, en los dos casos, es el mismo cambio en el ADN (Ver Figura 5.1).

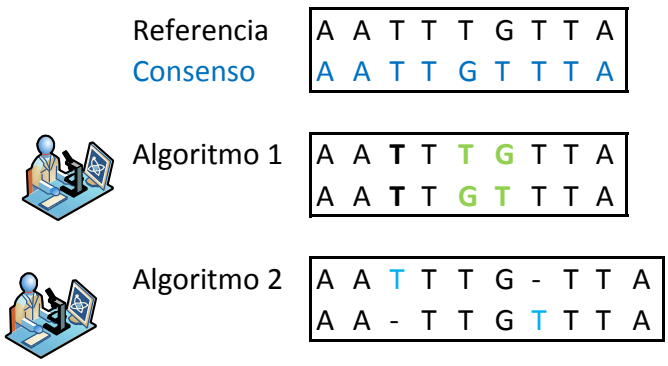


Figura 5.1: Problemas de los algoritmos de alineamiento

La solución que se propone para evitar este tipo de problemas en el marco de esta tesis es caracterizar las variaciones mediante regiones adyacentes, o flanking sequences, en lugar de identificarlas por posición.

Figura 5.2: Caracterización por flanking sequences

Las flanking sequences son el conjunto de nucleótidos que rodean una variación y son suficientemente precisos para caracterizarla. La documentación de una diferencia antes era ambigua (ejemplo 5.1), pero ahora queda acotada y no existe indeterminismo. En la Figura 5.2 la diferencia se puede documentar unívocamente mediante el tipo, Sustitución, el valor, GT, la fsizquierda, AATT, y la fsderecha TTA.

El problema ahora se traslada al hecho de averiguar si una flanking sequence es lo suficientemente precisa como para establecer una localización única en el genoma. Por lo tanto, si una diferencia se caracteriza por 20 nucleótidos por un extremo y 20 nucleótidos por el otro extremo, se elimina la ambigüedad y se asegura que no existe otra posición en el genoma que pueda tener las mismas regiones adyacentes. Se sabe que con 20 nucleótidos se identifica unívocamente una posición del genoma.

Un ser humano tiene, de media, una sustitución de un nucleótido respecto a la referencia cada 200 nucleótidos. Ya que una variación podría ocurrir en estos 20 nucleótidos, para asegurar que los resultados son completamente fiables, se utilizan las dos regiones adyacentes para la localización. De esta manera se permite una mayor variabilidad en los 20 nucleótidos de cada extremo.

Establecer esta metodología de caracterización evita todos los problemas de identificación de diferencias que proporcionan los algoritmos de alineamiento. Mediante esta caracterización, la búsqueda de diferencias ahora se puede realizar de manera dirigida, es decir, las variaciones que ya se conocen y están caracterizadas se buscan en la muestra y no, a la inversa, donde se buscan las diferencias y posteriormente se caracterizan.

Para la localización de diferencias se obtiene una lista de las variaciones ya caracterizadas que se van a intentar localizar en la muestra. Cada una de ellas dispone de dos flanking sequences, de manera que, para realizar la búsqueda, cada una de ellas se alinea contra la muestra. Una vez localizada la región del gen se comprueba si existe una diferencia en esa posición. Para ello se utiliza el valor de la variación y la otra flanking sequence.

Esta metodología, aunque ofrece resultados precisos, no permite el descubrimiento de nuevas diferencias, ya que, solo se buscan las variaciones que ya se conocen, y no existe forma de localizar una variación desconocida. En el contexto de esta tesis de máster, esta metodología

es viable, ya que, el soporte al diagnóstico consiste en corroborar el diagnóstico previo realizado por los médicos y, por lo tanto, consiste en localizar las variaciones que soporten dicho diagnóstico.

Sin embargo, se debe tener en cuenta que este caso no es siempre aplicable, en cuyo caso, para obtener todas las diferencias que existen en la muestra de un paciente, es necesario utilizar una metodología combinada.

5.2.3. Obtención de las secuencias de los dos alelos del ADN : La inferencia haplotípica

Las máquinas secuenciadoras solamente son capaces de obtener un electroferograma que representa los dos alelos del ADN. Cuando los dos alelos son idénticos, las señales que corresponden a cada alelo en el electroferograma quedan solapadas porque la señal de cada uno tiene el mismo valor. Cuando se producen variaciones en los dos alelos simultáneamente estas son identificables en los electroferogramas.

El problema se produce cuando los alelos son diferentes, es decir, existe una variación en heterocigosis. Si es una sustitución (en una posición, un alelo tiene un valor y el otro alelo un valor distinto), el problema se puede solventar debido a que las dos señales en ese punto son visibles. En la Figura 5.3, se puede distinguir que el quinto nucleótido es diferente en cada alelo, en un alelo el valor es A (color verde) y en el otro alelo el color es G (color negro).

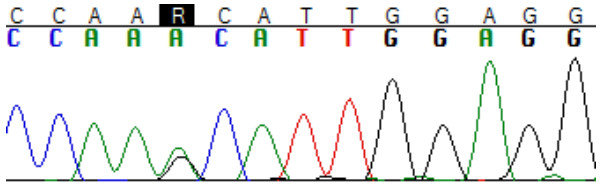


Figura 5.3: Electroferograma sustitución en heterocigosis

Sin embargo, este problema no se puede solventar cuando la variación es una inserción o una deleción. Cuando se produce este tipo de variaciones, se produce un desplazamiento de uno de los alelos. En el electroferograma aparecen las señales superpuestas de los dos alelos, por lo que no es posible distinguir qué valor corresponde a cada alelo en cada posición, ni que cambio se ha producido. En la Figura 5.4, se puede ver cuando se produce la variación: la señal es única hasta que aparecen dos señales diferentes en cada posición.

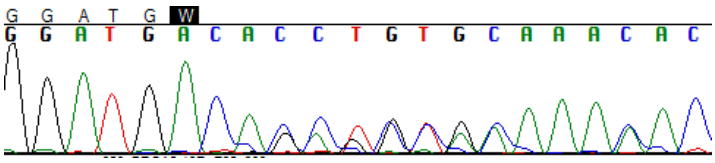


Figura 5.4: Electroferograma inserción/deleción en heterocigosis

Debido a que el análisis de ADN se realiza de manera textual, este problema también se manifiesta. Una variación de tipo sustitución en heterocigosis se expresa con un código de IUB, aquél que represente los dos valores. En cambio una variación de tipo inserción o deleción en heterocigosis se manifiesta porque desde el momento que se produce la variación hasta el final del segmento secuenciado toda la secuencia se expresa con códigos IUB, dando la sensación de que se producen sustituciones en todo el segmento, cuando en realidad lo que ha ocurrido es que se ha producido una única variación que ha provocado todo el desplazamiento.

Con el fin de solventar estos problemas, existen aproximaciones que analizan las señales obtenidas y aproximaciones que analizan los caracteres obtenidos a fin de obtener qué señal corresponde a cada alelo. Este problema se conoce, en la comunidad científica, como la inferencia haplotípica, ya que, se infiere cada uno de los alelos a partir de la señal solapada del electroferograma o los caracteres dobles de la secuencia obtenida.

La solución que se propone es utilizar la aproximación de la inferencia haplotípica basada en la secuencia de referencia [13]. Esta aproximación se basa en la idea de que si la variación se ha producido únicamente en un alelo, el otro alelo está alineado con la secuencia de referencia. La aproximación consiste en tres pasos:

1. Inferir el alelo que no ha cambiado: Mediante la referencia se obtiene la secuencia del alelo, ya que corresponde a los valores de la referencia.
2. Inferir el alelo que ha cambiado: Mediante la secuencia de la muestra y la secuencia del alelo previamente inferido, se obtiene la secuencia del otro alelo, ya que corresponde al otro valor de la muestra que no corresponde al alelo inferido.
3. Obtener la variación: Mediante el alineamiento de la secuencia inferida del alelo que ha cambiado con la referencia.

5.3. Especificación de Requisitos

En la fase de Alineamiento se plantean dos casos de usos para la obtención de diferencias entre la secuencia del paciente y la secuencia de referencia. Por un lado, se realiza la comparación entre secuencias, es decir, se buscan las diferencias que existen entre las dos secuencias. Este caso de uso además tiene en cuenta la comparación de secuencias en los casos en los que existe una variación en heterocigosis y, como consecuencia, la naturaleza de la secuencia objetivo cambia. Por otro lado, se realiza una búsqueda de diferencias a partir de una lista de variaciones. Este caso de uso, también tiene en cuenta la localización de diferencias cuando existe una variación en heterocigosis.

En la fase Alineamiento se llevan a cabo todas las tareas incluidas en el caso de uso "Obtener Diferencias" (ver Figura 5.5).

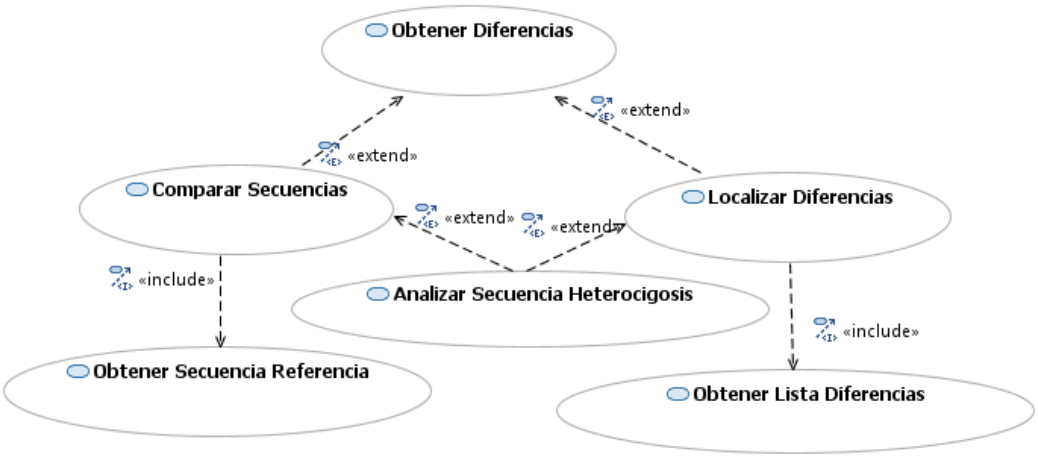


Figura 5.5: Caso de Uso "Obtener Diferencias"

Caso Uso "Obtener Diferencias"

El propósito del caso de uso es obtener una lista de diferencias que representen los cambios que se han encontrado en la secuencia objetivo respecto de la secuencia de referencia. Para ello se introducen las dos secuencias y se selecciona la aproximación que se quiere llevar a cabo para construir la lista de diferencias.

- Actores que intervienen: La HGDB proporciona información sobre la secuencia de referencia, y cuando la aproximación lo requiera, la lista de variaciones documentadas.
- Precondiciones: El gen que se está analizando ha sido seleccionado por el biólogo y, por lo tanto, es conocido en el sistema. La secuencia del paciente es correcta y completa.
- Postcondiciones: Se obtiene una lista de diferencias.
- Casos de uso que lo extienden: Comparar Secuencias y Localizar Diferencias.

En caso de uso está formado por los siguientes pasos:

1. El biólogo indica que aproximación quiere utilizar para obtener las diferencias. Existen dos opciones: 1) Comparar Secuencias; y 2) Localizar Diferencias.
2. Según la aproximación o aproximaciones seleccionadas en el paso anterior se ejecuta el caso de uso correspondiente.
3. El sistema devuelve al usuario la lista de diferencias obtenidas.

Caso Uso "Comparar Secuencias"

El propósito del caso de uso es obtener una lista de diferencias mediante el alineamiento y comparación de la secuencia objetivo respecto de la secuencia de referencia.

- Precondiciones: El gen que se está analizando ha sido seleccionado por el biólogo y, por lo tanto, es conocido en el sistema. La secuencia del paciente es correcta y completa.
- Postcondiciones: Se obtiene una lista de diferencias.
- Caso de uso que incluye: Obtener Secuencia Referencia
- Caso de uso que lo extiende: Analizar Secuencia Heterocigosis

En caso de uso está formado por los siguientes pasos:

1. El sistema obtiene la secuencia de referencia y la secuencia del paciente.
2. El sistema realiza un alineamiento en busca de diferencias para cada exón de la secuencia del paciente.
3. Si se da el caso, el sistema analiza la secuencia en heterocigosis y obtiene la diferencia.
4. El sistema devuelve al usuario la lista de diferencias obtenidas.

Caso Uso "Analizar Secuencia Heterocigosis"

El propósito del caso de uso es obtener una diferencia a partir del análisis de una secuencia donde se ha producido una variación en heterocigosis.

- Precondiciones: La secuencia objetivo tiene una variación en heterocigosis.
- Postcondiciones: Se obtiene una diferencia.
- Casos de uso que extiende: Comparar Secuencias y Localizar Diferencias

En caso de uso está formado por los siguientes pasos:

1. El sistema ejecuta el algoritmo para la identificación de la variación en heterocigosis.
2. El sistema devuelve al usuario la diferencia obtenida.

Caso Uso "Localizar Diferencias"

El propósito del caso de uso es obtener una lista de diferencias mediante la búsqueda de una lista de diferencias predefinidas en la secuencia del paciente.

- Precondiciones: El gen que se está analizando ha sido seleccionado por el biólogo y, por lo tanto, es conocido en el sistema. La secuencia del paciente es correcta y completa.
- Postcondiciones: Se obtiene una lista de diferencias.
- Caso de uso que incluye: Obtener Lista de Diferencias
- Caso de uso que lo extiende: Analizar Secuencia Heterocigosis

En caso de uso está formado por los siguientes pasos:

1. El sistema obtiene la secuencia del paciente y la lista de diferencias a buscar.
2. El sistema busca cada diferencia de la lista mediante el alineamiento de las flanking sequences con cada exón de la secuencia del paciente.
3. Si se da el caso, el sistema analiza la secuencia en heterocigosis y busca si alguna diferencia coincide con la variación en heterocigosis.
4. El sistema devuelve al usuario la lista de diferencias obtenidas.

Caso de Uso "Obtener Lista de Diferencias"

El propósito del caso de uso es obtener una lista de diferencias para buscar en la secuencia del paciente. Para ello, se obtiene el conocimiento del sistema de información genómico.

- Actores que intervienen: La HGDB proporciona información sobre las diferencias.
- Precondiciones: El gen que se está analizando ha sido seleccionado por el biólogo y por lo tanto es conocido en el sistema.
- Postcondiciones: Se obtiene la una lista de diferencias del gen.
- Caso de uso que lo incluye: Localizar Diferencias.

En caso de uso está formado por los siguientes pasos:

1. El sistema el realiza una petición a la HGDB indicando el gen seleccionado para el análisis.
2. La HGDB devuelve el conjunto de variaciones y sus propiedades del gen seleccionado.
3. El sistema devuelve una lista de diferencias.

5.4. Modelado Conceptual

La fase de Alineamiento es la responsable de obtener las diferencias de la secuencia genética del paciente respecto a la secuencia de referencia.

5.4.1. Detección de diferencias de la secuencia de ADN

Para llevar a cabo la detección de diferencias entre dos secuencias de ADN se utilizan los algoritmos de alineamiento que se encargan de obtener la lista de diferencias entre las dos secuencias.

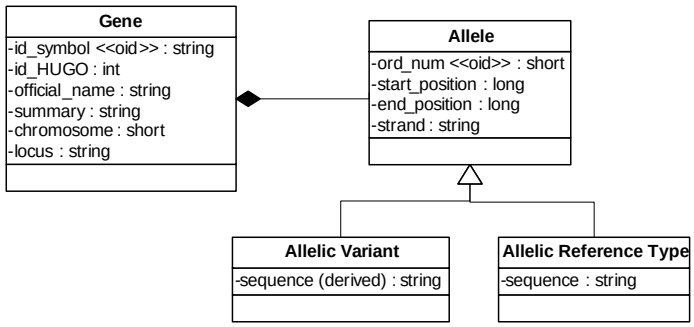


Figura 5.6: Esquema Conceptual del Genoma Humano (CSHG)

El modelo conceptual para la ejecución del alineamiento no tiene una asociación directa con el CSHG, ya que se trata de un análisis textual con el objetivo de obtener diferencias entre secuencias de texto. Sin embargo, al tratarse de algoritmos de alineamiento genéticos, es decir, algoritmos que realizan una análisis de cadenas de texto que representan secuencias genéticas, si que se extrae conocimiento del sistema de información genómico. La secuencia de Referencia que se utiliza en la comparación es la entidad *Allelic Reference Type* de *Gene* (Figura 5.6).

Debido a la naturaleza del análisis, el modelo conceptual mostrado en la Figura 5.7, combina los conceptos de un alineamiento de cadenas de caracteres, con las propiedades biológicas de las secuencias genéticas. Por esa razón, tanto las cadenas de texto, como las diferencias obtenidas, adquieren propiedades de este tipo de secuencias. La entidad conceptual *Alineamiento* representa la comparación entre varias cadenas de caracteres. La *Secuencia de Referencia*, es la secuencia base sobre la que se realiza la comparativa. Tiene un atributo *secuencia* que contiene la cadena de caracteres que que representa el formato textual de la *Secuencia de Referencia*. La entidad *Secuencia Objetivo* es la secuencia que quiere compararse con la *Secuencia de Referencia*. De la misma manera, tiene un atributo *secuencia* que contiene la cadena de caracteres que representa el formato textual de la *Secuencia Objetivo*. La *Secuencia Objetivo* puede representar una secuencia genética en la que los dos alelos están desplazados. Si esto ocurre, en un punto de la secuencia objetivo todos los caracteres reflejan el desplazamiento. Esta propiedad de la secuencia objetivo se representa mediante el atributo *desplazada*.

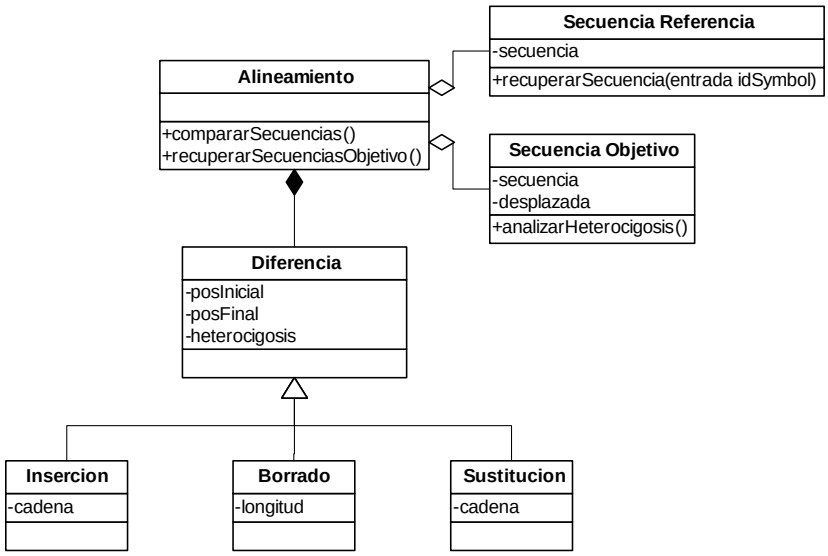


Figura 5.7: Modelo Conceptual Detección Diferencias

La entidad conceptual *Diferencia* modela como difiere una *Secuencia Objetivo* de la *Secuencia* de *Referencia*. Una diferencia se expresa mediante una posición inicial, *posInicial*, y una posición final, *posFinal*, que sitúan la diferencia respecto a la *Secuencia de Referencia*. El atributo *heterocigosis* representa una diferencia localizada en una *Secuencia Objetivo* desplazada. Una diferencia puede ser de tres tipos: Inserción, Borrado o Sustitución. Una *Diferencia* es una *Inserción* cuando se han añadido uno a varios caracteres en la *Secuencia Objetivo* respecto a la *Secuencia de Referencia*. El atributo *cadena* contiene el valor de los caracteres añadidos. Una *Diferencia* es un *Borrado* cuando se han eliminado uno a varios caracteres en la *Secuencia Objetivo* respecto a la *Secuencia de Referencia*. El atributo *longitud* contiene el numero de caracteres eliminados. Una *Diferencia* es una *Sustitución* cuando uno a varios caracteres han cambiado en la *Secuencia Objetivo* respecto a la *Secuencia de Referencia*. El atributo *cadena* contiene el valor de los caracteres modificados.

Para llevar a cabo la fase de alineamiento y proporcionar la funcionalidad expresada en los casos de uso, es necesario proveer a las clases de métodos que ejecuten la funcionalidad necesaria (Figura 5.7).

Para llevar a cabo el caso de uso "Comparar Secuencias" es necesario crear una instancia de la clase Alineamiento y ejecutar el método compararSecuencias:

Caso Uso "Comparar Secuencias" Se crea una instancia de la Secuencia de Referencia y se obtiene la información de la secuencia mediante el método *recuperarSecuencia(entrada idSymbol)*. A continuación, se crean las instancias necesarias de la Secuencia Objetivo, una por cada exón de la secuencia, mediante el método *recuperarSecuenciasObjetivo()*. Una vez creadas todas las entidades necesarias para ejecutar el alineamiento, la entidad Alineamiento proporciona el método Comparar Secuencias:

compararSecuencias()

Este método se encarga de obtener la lista de diferencias mediante la ejecución del algoritmo de alineamiento y la interpretación de sus resultados. El método se encarga de:

1. Ejecutar el algoritmo de alineamiento para cada una de las Secuencias Objetivo. Si se trata de una secuencia desplazada, se realiza una llamada al método *analizarHeterocigosis()*.
2. Recorrer el alineamiento e interpretar las diferencias obtenidas.
3. Crear la instancia de Diferencia que corresponda: Inserción, Borrado o Sustitución y proporcionar la información asociada.

5.4.2. Búsqueda Dirigida de Diferencias de la Secuencia de ADN

Para llevar a cabo una búsqueda diferencias en una secuencia de ADN a partir de una lista de diferencias predeterminada se utilizan los algoritmos de alineamiento. Estos algoritmos localizan las regiones adyacentes de cada diferencia en la secuencia genética del individuo. Como resultado se obtiene un subconjunto de la lista de diferencias original, es decir, solo aquéllas que se han localizado en la secuencia.

El modelo conceptual diseñado para la ejecución de la búsqueda está basado en una serie de conceptos del CSHG (Figura 5.8). La búsqueda dirigida de diferencias obtiene una lista de diferencias objetivo desde el sistema de información genómico. Para ello, se establece una relación entre el concepto de variación genómica, entidad conceptual *Variation*, y el concepto de diferencia. De ese modo, una lista de *Variations* se transforma a lista de *Diferencias* eliminando el componente genético e incorporando el componente textual. Así, una *Variation* de tipo *Insertion*, se convierte en una *Diferencia* de tipo *Insercion*, una *Variation* de tipo *Deletion* se convierte en una *Diferencia* de tipo *Borrado* y una *Variation* de tipo *Indel* se convierte en una *Diferencia* de tipo *Sustitucion*. Además se utiliza el concepto *Allelic Reference Type*, que representa la *Secuencia Referencia* del modelo, debido a que las variaciones extraídas del sistema genómico están documentadas respecto a la secuencia de referencia *Allelic Reference Type*.

El modelo conceptual diseñado (Figura 5.9) refleja una serie de pequeños cambios respecto al modelo conceptual utilizado para la detección de diferencias. Se utilizan las mismas entidades conceptuales, ya que se trata de la misma fase, pero se modifican una serie de atributos que permiten definir correctamente los conceptos involucrados en la búsqueda. La entidad *Diferencia*, ahora se caracteriza mediante los atributos *fsderecha* y *fsizquierda* que representan las dos regiones de nucleótidos de la *Secuencia Referencia* que rodean el cambio. El atributo *encontrada* indica que la diferencia se ha encontrado en alguna de las *Secuencias Objetivo*, y el atributo *heterocigosis* que la diferencia se ha encontrado en una *Secuencia Objetivo* desplazada.

En este caso las *Diferencias* se intentan localizar en la *Secuencias Objetivo* mediante la ejecución del alineamiento entre las flanking sequences, *fsderecha* y *fsizquierda*, de las *Diferencias*

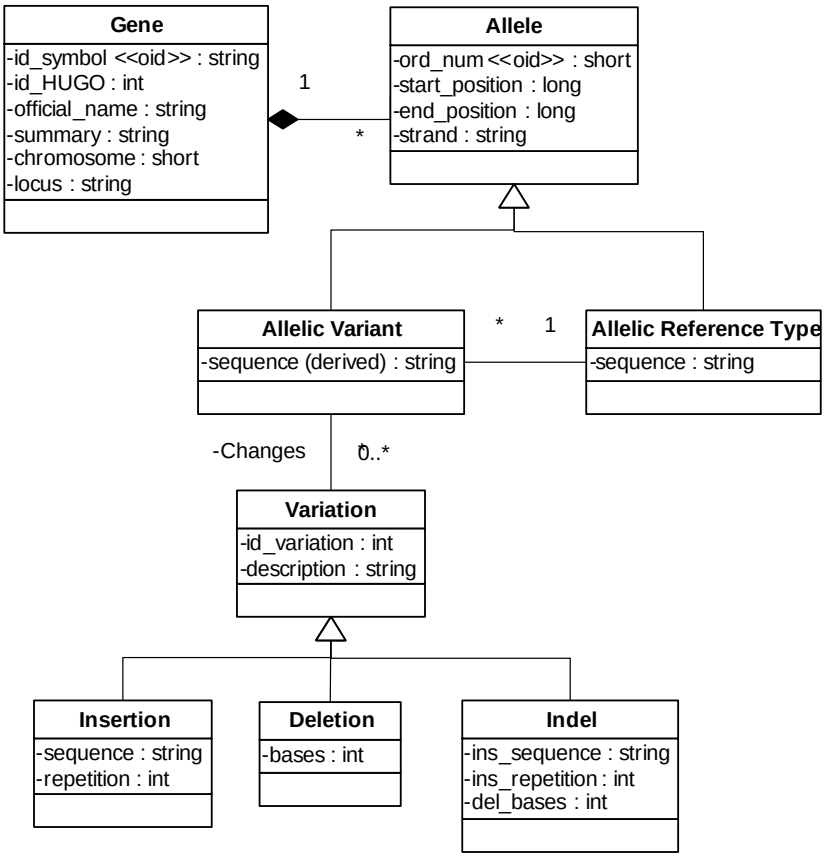


Figura 5.8: Esquema Conceptual del Genoma Humano (CSHG)

y la *secuencia* de la *Secuencia Objetivo*. Aunque son estas dos entidades las que intervienen en la ejecución del alineamiento, conceptualmente, las *Diferencias* representan información obtenida de un *Alineamiento* entre la *Secuencia* de *Referencia* y una *Secuencia Objetivo,* por lo que el modelo conceptual no cambia.

Para llevar a cabo la fase de alineamiento y proporcionar la funcionalidad expresada en los casos de uso, es necesario proveer a las clases de métodos que ejecuten la funcionalidad necesaria (Figura 5.9).

Para llevar a cabo el caso de uso "Localizar Diferencias" es necesario crear una instancia de la clase Alineamiento y ejecutar el método buscarDiferencias:

Caso Uso "Localizar Diferencias" Se crean las instancias necesarias de la Secuencia Objetivo, una por cada exón de la secuencia, mediante el método *recuperarSecuenciasObjetivo()* y las instancias necesarias de Diferencias, obtenidas del sistema de información genómico, mediante el método *recuperarDiferencias()*. Una vez creadas todas las entidades necesarias para ejecutar el alineamiento, la entidad Alineamiento proporciona el método Buscar Diferencias:

buscarDiferencias()

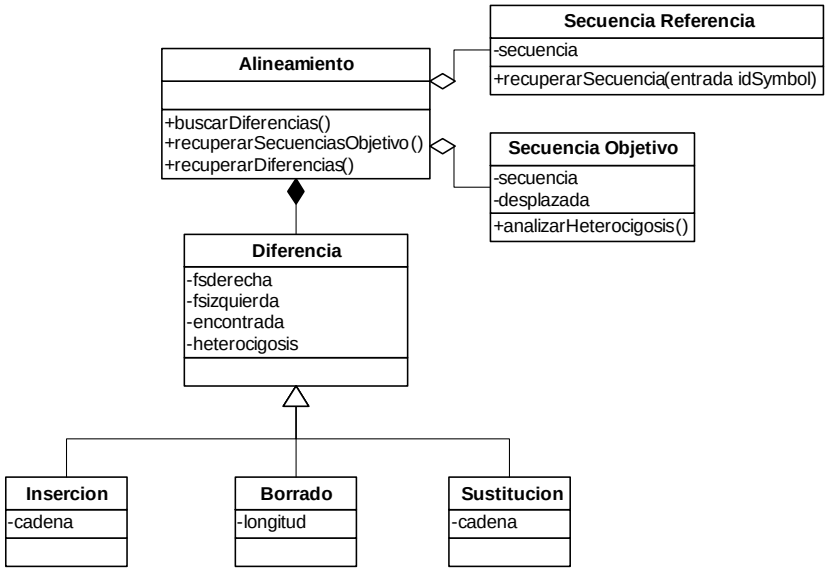


Figura 5.9: Modelo Conceptual Búsqueda Dirigida de Diferencias

Este método obtiene la lista de diferencias que se encuentran en la secuencia objetivo mediante la ejecución del algoritmo de alineamiento y la interpretación de sus resultados. El método se encarga de:

1. Ejecutar el algoritmo de alineamiento para cada una de las *Diferencias* y cada una de las Secuencias Objetivo. Si se trata de una secuencia desplazada se realiza una llamada al método *analizarHeterocigosis()*.

2. Si se se ha encontrado coincidencia de las flanking sequences y el tipo de Diferencia concuerda (Insercion, Borrado o Sustitucion), la Diferencia se considera encontrada. Si además la Secuencia Objetivo era una secuencia desplazada, la Diferencia está en heterocigosis.

5.5. Implementación

El desarrollo de la fase se ha llevado a cabo mediante la integración de tres herramientas de alineamiento (Figura 5.10). En primer lugar, se ha integrado la implementación del algoritmo Blast proporcionada por NCBI (BLAST), y en segundo lugar, se han integrado dos herramientas desarrolladas en el marco del proyecto Diagen, llamadas SmithWatterman (SW) y FlankingSequences (Flanking). Las herramientas SW y Flanking se han implementado junto con la colaboración del grupo de investigación GryCap perteneciente a la Universidad Politécnica de Valencia. El algoritmo de alineamiento debe su nombre al hecho de estar basado en el algoritmo Smith-Watterman [36]. En la implementación de este nuevo algoritmo se aplican las soluciones propuestas en este capítulo para obtener mayor precisión en los resultados y para la detección de diferencias en heterocigosis.

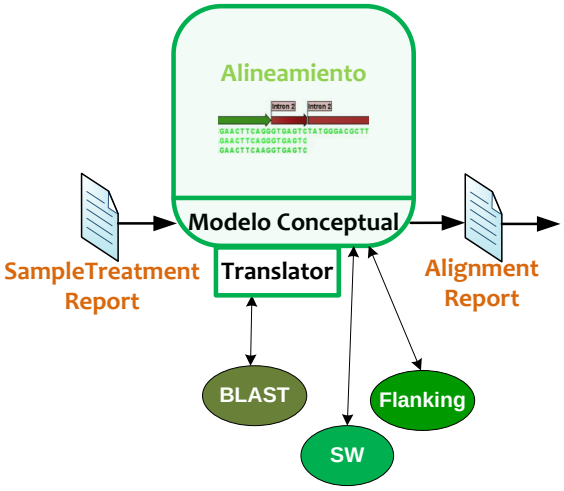


Figura 5.10: Fase Alineamiento

En primer lugar, para conseguir la integración de la herramienta BLAST, se desarrolla un traductor que transforma la entrada expresada en términos del modelo conceptual en las entradas de BLAST y las salidas en el contexto de Blast en términos del modelo conceptual.

En segundo lugar, para conseguir la integración de las herramientas SW y Flanking, no es necesario implementar ningún traductor. La implementación llevada a cabo utiliza el modelo conceptual de la fase de Alineamiento, por lo que no requiere una traducción de la entrada ni una traducción de la salida producida.

5.5.1. Modelo Conceptual Reporte de Resultados

Se ha diseñado un modelo conceptual de Reporte de Resultados, *AlignmentReport*, para expresar los resultados de manera precisa o, en caso de utilizarse en el contexto de otra herramienta, para guiar el proceso de flujo de datos.

El modelo conceptual de la Figura 5.11 representa las diferencias que existen en una secuencia genética respecto a una secuencia genética de referencia. Este modelo es una vista del modelo conceptual de la fase Alineamiento. La entidad conceptual *AlignmentReport* representa el reporte generado después de realizar el alineamiento para la obtención de diferencias de la muestra de ADN del paciente, y el atributo *geneId* contiene el identificador del gen de dicho reporte.

En la Figura 5.12 se muestra como se integran las tres herramientas en el framework. Para la integración de Blast se implementando dos módulos, el módulo Output y el módulo Input. Mediante el BlastInputTranslator se transforma la secuencia genética expresada mediante el modelo conceptual SampleTreatmentReport. De este modo la herramienta Blast puede leer la secuencia genética en el formato específico de su implementación. Mediante el BlastOutputTranslator se transforma la salida producida por Blast en términos del modelo conceptual.

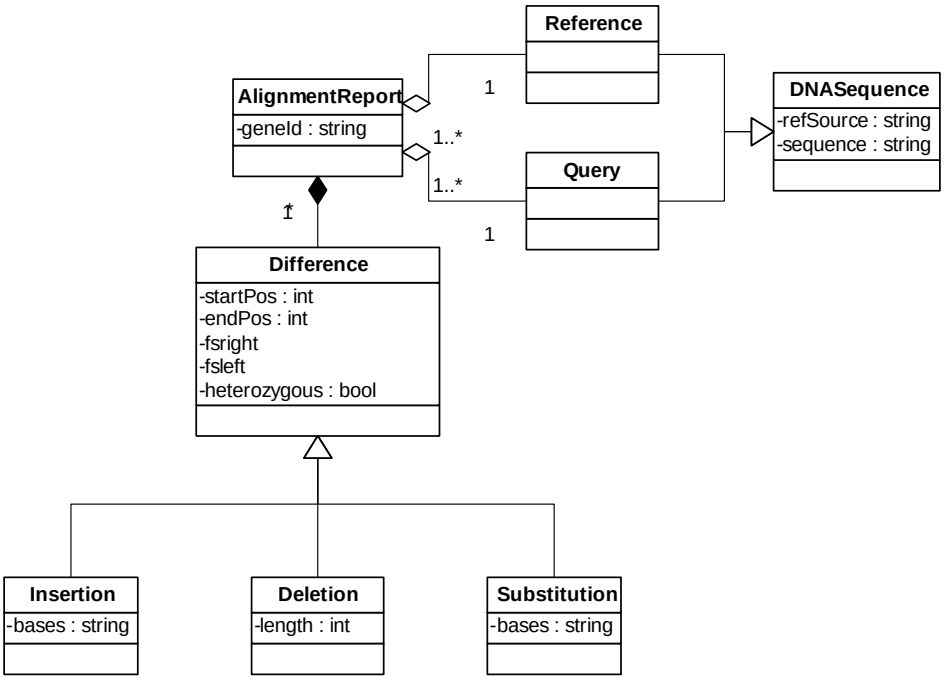


Figura 5.11: Modelo Conceptual AlignmentReport

Para la integración de SW y Flanking no es necesario implementar el módulo Input ni el módulo Output.

5.5.2. Implementación del Modelo Conceptual Reporte de Resultados

La implementación de la fase de Alineamiento se ha desarrollado en Java y el modelo conceptual, de la misma manera que el SampleTreatmentReport, se ha implementado mediante clases Java y mediante la especificación XMLSchema.

Una vez implementada la funcionalidad y los artefactos necesarios para expresar los resultados en términos del modelo conceptual, la fase de Alineamiento puede ejecutarse de manera independiente. En la Figura 5.13 se puede observar un ejemplo de la salida XML que proporciona la ejecución la fase de Alineamiento de manera independiente. Para ello se ha utilizado una muestra del gen BRCA2, en la que se han localizado 10 diferencias con respecto a la secuencia de referencia NG_012772.1.

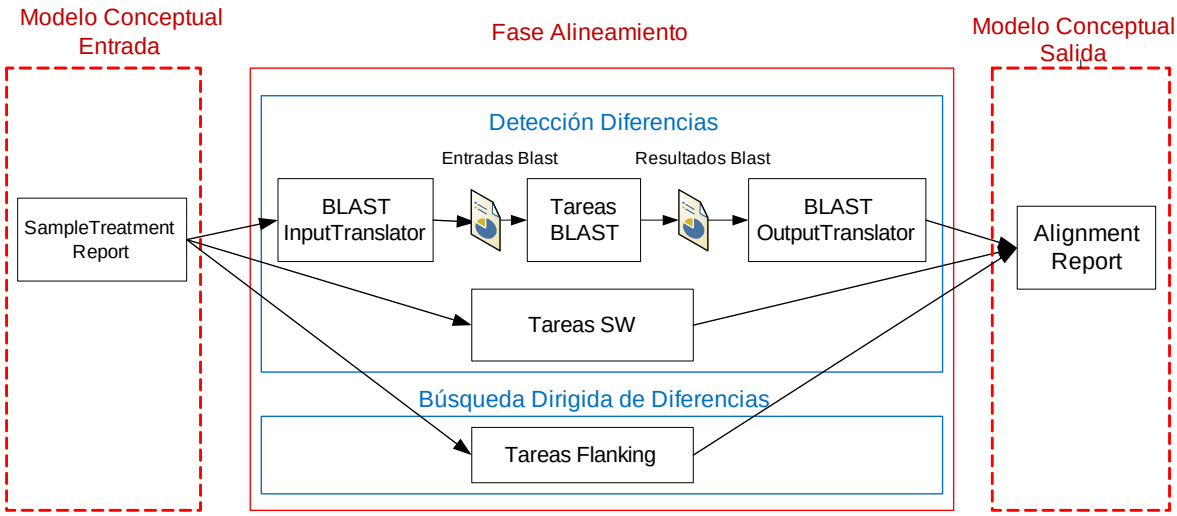


Figura 5.12: Integración de la utilidad Blast de NCBI y algoritmo de SmithWatterman

```
<Alignment geneId="BRCA2">
    <Reference><refsource>NG_012772.1</refsource></Reference>
    <Consensus><refsource>Query.txt</refsource></Consensus>
    <Differences>
        <Substitution endPos="5956" initialPos="5956"
            fsl="TTTGCAGACTTATTTACCAA" fsr="CATTGGAGGAATATCGTAGG"
            heterozygosis="true">A</Substitution>
        <Substitution endPos="27683" initialPos="27683"
            fsl="AGTAAATGTCATGATTCTGT" fsr="GTTTCAATGTTTAAGATAGA"
            heterozygosis="true">C</Substitution>
        <Substitution endPos="28439" initialPos="28439"
            fsl="AAGATCAAAGAACCTACTCT" fsr="TTGGGTTTTCATACAGCTAG"
            heterozygosis="false">G</Substitution>
        <Substitution endPos="30389" initialPos="30389"
            fsl="TTGGTATTAGGAACCAAAGT" fsr="TCACTTGTTGAGAACATTCA"
            heterozygosis="false">C</Substitution>
        <Substitution endPos="36228" initialPos="36228"
            fsl="GTTAACATTTATTGAGCATC" fsr="GTTACATTCACTGAAAATTG"
            heterozygosis="false">C</Substitution>
        <Substitution endPos="44616" initialPos="44616"
            fsl="CCACCTTTTAAAACTAAATC" fsr="CATTTTCACAGAGTTGAACA"
            heterozygosis="true">G</Substitution>
        <Substitution endPos="46320" initialPos="46320"
            fsl="TGCTAGGCCTGCCTCATCCT" fsr="CTAAAGTGATCTGTGCTTCC"
            heterozygosis="false">A</Substitution>
        <Substitution endPos="68772" initialPos="68772"
            fsl="TTGTCCTGTTTAAAGCCATC" fsr="AGTTACAATAGATGGAACTT"
            heterozygosis="true">C</Substitution>
        <Substitution endPos="22113" initialPos="22113"
            fsl="CATTAGATTCAAATGTAGCA" fsr="ATCAGAAGCCCTTTGAGAGT"
            heterozygosis="true">C</Substitution>
        <Substitution endPos="52030" initialPos="52030"
            fsl="TTTTTATGATAATATTCTAC" fsr="TTTATTTGTTCAGGGCTCTG"
            heterozygosis="true">C</Substitution>
    </Differences>
</Alignment>
```

Figura 5.13: Ejemplo XML Alignment Report

Capítulo 6

Fase Conocimiento

La fase Conocimiento es la fase que proporciona información de diferencias localizadas en una secuencia genética, a partir de la explotación de un sistema genómico. Para llevar a cabo este proceso de datos es necesario caracterizar las diferencias como variaciones genéticas y posteriormente extraer el conocimiento asociado a las variaciones. Mediante este conocimiento se puede averiguar si la variación ya ha sido documentada anteriormente por algún biólogo en el dominio. En ese caso, se conoce si la variación tiene un efecto patológico, y se obtener la bibliografía asociada, es decir, los artículos que hablan de la variación en relación a una enfermedad.

En primer lugar, se proporciona un conjunto de conceptos biológicos para comprender que información es necesaria para caracterizar una diferencia y como esta información se usa para el soporte al diagnóstico.

En segundo lugar, se identifican los problemas encontrados a la hora de realizar la búsqueda de variaciones genéticas en diferentes repositorios y las soluciones que se adoptan para solventarlas. En el diseño de esta fase se abordan los problemas de heterogeneidad de los sistemas de información: 1) Búsqueda manual en diferentes repositorios genéticos; y 2) Ambigüedad en la caracterización de variaciones genéticas.

En tercer lugar, se plantean los diferentes casos de uso necesarios para caracterización de diferencias y, concretamente, para la localización de variaciones genéticas en el sistema de información genómico HGDB. La localización de variaciones se puede realizar por posición o mediante regiones adyacentes.

En cuarto lugar, se explica el modelo conceptual diseñado y su correspondencia con el CSHG. Se presentan las entidades conceptuales y los métodos necesarios para llevar a cabo todos los casos de uso planteados para la caracterización de las diferencias encontradas en la secuencia genética de un paciente.

Por último, se explica la implementación llevada a cabo y el modelo conceptual de Reporte de Resultados *VariationKnowledgeReport*, utilizado para expresar los resultados de la fase de Conocimiento de manera precisa.

6.1. Background Biológico

Las variaciones genéticas

Una variación genética representa la existencia de uno o varios nucleótidos en el ADN de un ser humano que se considera que no es compartido por todos los seres humanos.

Las variaciones genéticas se clasifican en tres tipos dependiendo de la naturaleza del cambio:

1. Inserciones: Se insertan uno o varios nucleótidos en el ADN.
2. Deleciones: Se eliminan uno o varios nucleótidos en el ADN.
3. Indels: Se modifican uno o varios nucleótidos en el ADN.

En ocasiones, estas variaciones afectan a la fabricación de proteínas provocando un cambio. Este cambio de proteínas puede no tener efecto, o producir un cambio significativo en el fenotipo. Un cambio fenotípico significativo puede derivar, simplemente, una característica externa diferente, lo que explica las diferencias que tenemos todos los seres humanos entre nosotros mismos, o incluso entre diferentes sexos y razas. Específicamente, la variación se denomina SNP si se trata de una variación de tipo indel de un solo nucleótido y esta variación se dan en al menos 1 % de la población. Si una variación tiene un fenotipo perjudicial para la salud entonces se denomina mutación.

Asociación genotipo-fenotipo y el soporte al diagnóstico

Los investigadores genéticos se encargan de buscar la explicación de los fenotipos dañinos para los seres humanos. Para ello, investigan los genotipos de los individuos que padecen ciertas enfermedades, cuyo origen puede ser genético.

Cuando un conjunto de individuos padecen una enfermedad genética, se estudian sus genotipos con la finalidad de obtener resultados que respalden que un genotipo concreto está provocando la enfermedad de estudio. Cuando se obtienen resultados precisos, esta asociación genotipo-fenotipo se publica de manera que la comunidad científica tenga conocimiento de los descubrimientos obtenidos.

De este modo, para diagnosticar una enfermedad de origen genético se buscan las variaciones en la secuencia genética y se consultan las publicaciones de la comunidad científica con la finalidad de encontrar una que informe si alguna variación localizada en un paciente provoca la enfermedad genética.

6.2. Problemas detectados y Soluciones

Con el objetivo de mostrar una lista de variaciones genéticas, el fenotipo asociado y la bibliografía asociada, es necesario resolver los problemas actuales de heterogeneidad de las fuentes de datos genómicos.

La solución a este problema ha sido propuesta en el marco del proyecto Diagen. La elaboración de un sistema de información homogéneo, basado en el esquema conceptual del genoma humano, y la carga de datos estructurados a partir de diferentes fuentes de datos genómicas.

Esta solución, es capaz de solucionar los problemas de heterogeneidad planteados:

- Obligatoriedad de conocer el funcionamiento de cada fuente de manera independiente: Se evita la necesidad de conocer múltiples formas de expresar, buscar y obtener la caracterización de cada diferencia para cada fuente de datos genómicos.
- Diversidad de formatos: Una variación se expresa en términos del modelo conceptual. Ya no es necesario expresar las variaciones en diferentes formatos textuales como por ejemplo: 1000delAAA, Deletion 1000_1004, Del1000_1004.
- Procesamiento manual en la caracterización, búsqueda y recuperación de variaciones: Debido a la homogeneidad del esquema conceptual estas operaciones se pueden automatizar.

Sin embargo, a la hora de obtener el conocimiento del sistema de información genómico, se observa que las fuentes genómicas, de las cuales se ha importado los datos, arrastran errores influenciados por los algoritmos de alineamiento, errores ya detectados en la fase anterior. El problema reside en la caracterización ambigua de las variaciones, hasta ahora caracterizadas por posición. Si las posiciones obtenidas varían en función del algoritmo de alineamiento, pero son correctas, no es posible solucionar este problema en el contexto de la carga de datos, sino que debe solucionarse en el contexto de la explotación de datos.

Para solucionar este problema se aplica las misma solución que en la fase anterior, utilizar las regiones adyacentes para localizar una variación. De ese modo, cada una de las diferencias obtenidas en la fase de alineamiento se pueden buscar por posición o por flanking sequences.

6.3. Especificación de Requisitos

En la fase de Conocimiento se plantean los siguientes casos de uso para analizar la lista de diferencias obtenidas en la fase de Alineamiento (Figura 6.1). El análisis de diferencias se lleva a cabo mediante la caracterización de cada una de ellas como variaciones genéticas. Esta caracterización consiste en: 1) Eexpresar las diferencias como variaciones; 2) Buscarlas en el sistema de información genómica para caracterizarlas como conocidas o no conocidas; y 3) Localizar el fenotipo y la bibliografía asociada. La búsqueda de variaciones puede realizarse de dos modos, mediante la localización por posición o mediante la localización por similitud de las regiones adyacentes o flanking sequences.

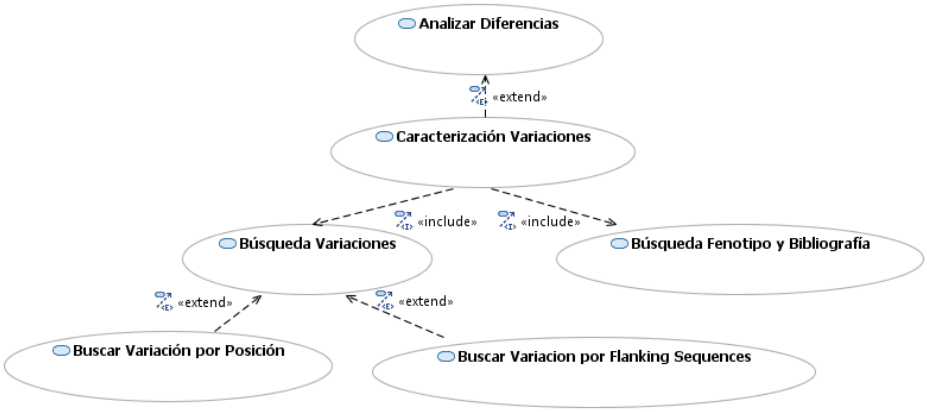


Figura 6.1: Caso Uso "Analizar Diferencias"

Caso Uso "Analizar Diferencias"

El propósito del caso de uso es obtener información sobre una lista de diferencias. Existen diferentes tipos de conocimiento que se puede obtener de a partir de una lista de diferencias. En el marco de esta tesis de máster, se proporciona la caracterización de variaciones para obtener la información fenotípica que ofrezcan soporte a la detección de enfermedades.

- Actores que intervienen: La HGDB proporciona conocimiento sobre las variaciones. El biólogo proporciona información sobre el análisis de las diferencias que desea realizar.
- Precondiciones: El gen que se está analizando ha sido seleccionado por el biólogo y, por lo tanto, es conocido en el sistema.
- Postcondiciones: Se obtiene una lista de variaciones caracterizadas fenotípicamente.
- Casos de uso que lo extienden: Caracterización Variaciones.

En caso de uso está formado por los siguientes pasos:

1. El sistema analiza las diferencias mediante la ejecución de la caracterización fenotípica (En caso de existir otro tipo de análisis, el biólogo puede seleccionar el caso de uso).
2. El sistema devuelve la lista de variaciones caracterizadas fenotípicamente.

Caso Uso "Caracterización Variaciones"

El propósito del caso de uso es caracterizar cada una de las diferencias. Para ello, se obtiene el tipo de variación, si esta es conocida y qué fenotipo tiene asociado.

- Actores que intervienen: La HGDB proporciona conocimiento sobre las variaciones. El biólogo proporciona información sobre el análisis de las diferencias que desea realizar.
- Precondiciones: El gen que se está analizando ha sido seleccionado por el biólogo y, por lo tanto, es conocido en el sistema.
- Postcondiciones: Se obtiene una lista de variaciones caracterizadas fenotípicamente.
- Casos de uso que incluye: Búsqueda Variaciones y Búsqueda Fenotipo y Bibliografía.

En caso de uso está formado por los siguientes pasos:

1. El sistema expresa cada una de las diferencias como variaciones genéticas.
2. El sistema realiza la búsqueda de las variaciones genéticas.
3. El sistema realiza la búsqueda del fenotipo y la bibliografía de las variaciones genéticas.
4. El sistema devuelve la lista de variaciones caracterizadas fenotípicamente.

Caso Uso "Búsqueda Variaciones"

El propósito del caso de uso es caracterizar cada una de las variaciones como conocida o desconocida según se hayan localizado en el sistema de información genómico.

- Actores que intervienen: La HGDB proporciona conocimiento sobre las variaciones. El biólogo proporciona información sobre el análisis de las diferencias que desea realizar.
- Postcondiciones: Se obtiene una lista de variaciones localizadas.
- Caso de uso que lo incluye: Caracterización Variaciones.
- Casos de uso que lo extienden: Búsqueda Variaciones por Posición y Búsqueda Variaciones por Flanking.

En caso de uso está formado por los siguientes pasos:

1. El biólogo selecciona que tipo de búsqueda desea realizar. Existen dos opciones: 1) Búsqueda por posición; y 2) Búsqueda por flanking sequences.
2. El sistema ejecuta la búsqueda seleccionada por el biólogo.
3. El sistema devuelve la lista de variaciones indicando cuales son conocidas.

Caso Uso "Buscar Variación por Posición"

El propósito del caso de uso es localizar una variación en el sistema de información genómico buscándola por posición.

- Actores que intervienen: La HGDB proporciona conocimiento sobre las variaciones.
- Postcondiciones: Se obtiene si la variación está en la HGDB o no.
- Casos de uso que extiende: Búsqueda Variaciones.

En caso de uso está formado por los siguientes pasos:

1. El sistema realiza una consulta a la HGDB proporcionando la información de una variación donde se incluye la posición a buscar.
2. La HGDB responde a la consulta indicando si la variación es conocida en el sistema o no.

Caso Uso "Buscar Variación por Flanking Sequences"

El propósito del caso de uso es localizar una variación en el sistema de información genómico buscándola por las flanking sequences.

- Actores que intervienen: La HGDB proporciona conocimiento sobre las variaciones.
- Postcondiciones: Se obtiene si la variación está en la HGDB o no.
- Casos de uso que extiende: Búsqueda Variaciones.

En caso de uso está formado por los siguientes pasos:

1. El sistema realiza una consulta a la HGDB proporcionando la información de una variación donde se incluye las flanking sequences a buscar, así como el grado de similitud que se espera encontrar.
2. La HGDB responde a la consulta indicando si la variación es conocida en el sistema o no.

Caso Uso "Búsqueda Fenotipo y Bibliografía"

El propósito del caso de uso es caracterizar cada una de las variaciones mediante su fenotipo asociado y la bibliografía que respalda la asociación con dicho fenotipo.

- Actores que intervienen: La HGDB proporciona conocimiento sobre las variaciones.
- Postcondiciones: Se obtiene una lista de variaciones localizadas.
- Caso de uso que lo incluye: Caracterización Variaciones.

En caso de uso está formado por los siguientes pasos:

1. El sistema realiza una consulta a la HGDB proporcionando la información de una variación.
2. La HGDB devuelve la información asociada a la variación, es decir, fenotipo y bibliografía.
3. El sistema devuelve la lista de variaciones con sus fenotipos y bibliografía.

6.4. Modelado Conceptual

La fase de Conocimiento es la responsable de extraer información sobre las diferencias encontradas en una secuencia de ADN en modo de variaciones genéticas.

En la Figura 6.2 se muestra la vista del CSHG que contiene los conceptos biológicos involucrados en la ejecución de esta fase. Las entidades *Gene*, *Allele*, *AlellicVariant, SplicedTranscript, mRNA, Others RNA, ElementTranscript, Exon, Intron, Variation, Insertion, Deletion e Indel,* han sido explicadas en los capítulos anteriores. En esta vista, un *Indel* es de tipo *SNP* si es un indel de un solo nucleótido y además se manifiesta en al menos 1 % de la población.

La entidad *Phenotype* modela las características externas que se manifiestan en el ser humano. En el marco de este proyecto, estas características se refieren enfermedades de origen genético. El atributo *id_syndrome* es un identificador del fenotipo para identificar distintas instancias y el atributo *name* es una descripción del fenotipo. La asociación entre la aparición de una *Variation* y, como consecuencia, la mani-festación de un *Phenotype,* se modela mediante la relación con la entidad *Certaninty*, que representa la certeza con la que se asegura que existe la relación de causa-efecto. El grado de certeza se indica mediante el atributo *level.*

La entidad *Bibliography Reference* modela cada una de las publicaciones académicas que tratan de temas relacionados con el genoma humano, y concretamente se relaciona con la entidad Variation si la publicación trata sobre ella. Los atributos *title*, *author*, *abstract*, *publication* y *date* representan respectivamente el título, la lista de autores, un pequeño resumen de la

publicación, el congreso, revista, libro, etc, donde se ha publicado y la fecha de publicación. La entidad *Bibliography DB* modela el banco de datos que almacena dicha publicación.

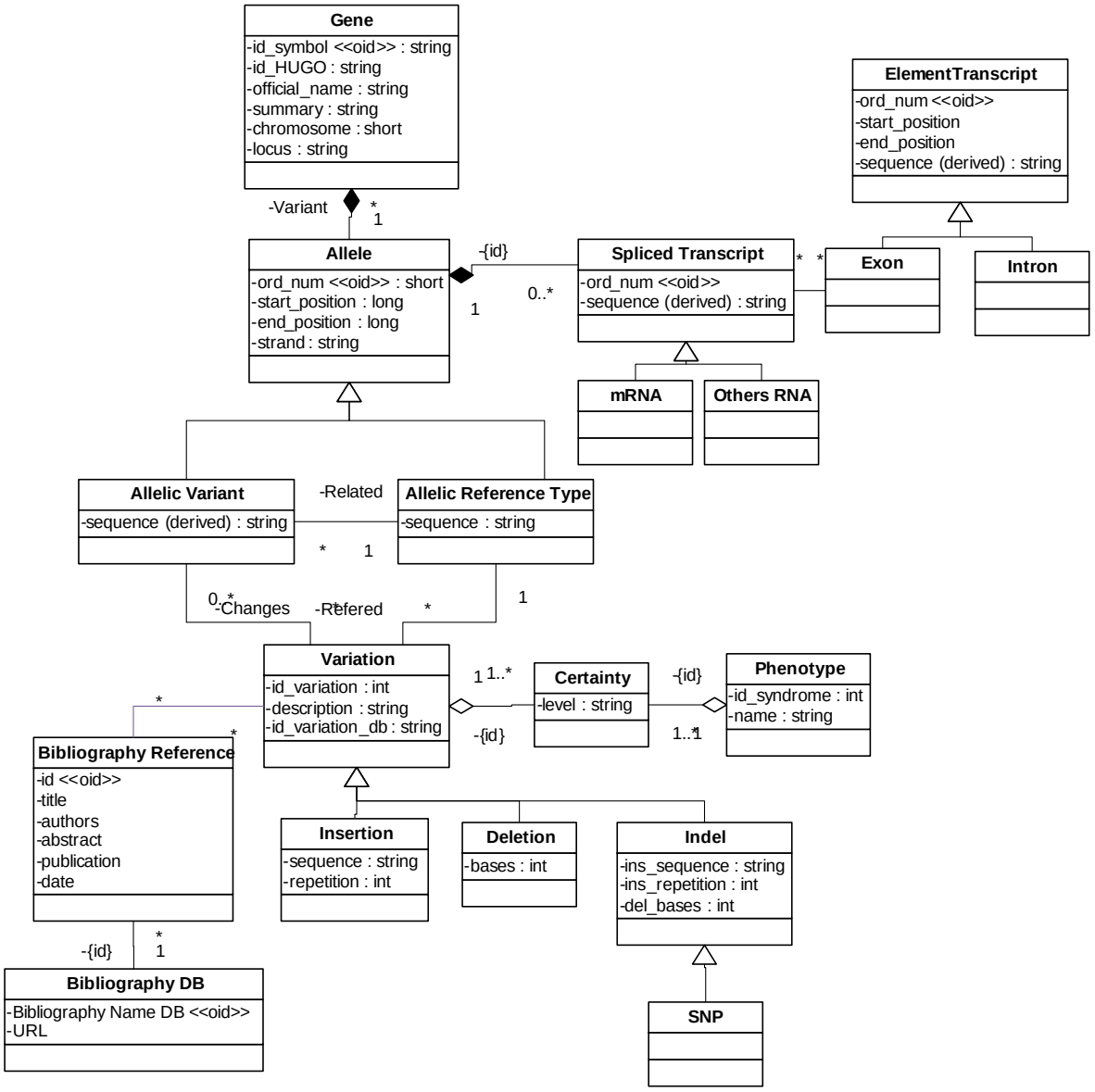


Figura 6.2: Esquema Conceptual del Genoma Humano

El modelado para la extracción de conocimiento para esta fase está basada en esta vista del CSHG (Figura 6.3). La entidad conceptual *Gene* recoge todos los conceptos relacionados con el gen del análisis, tanto las propiedades del concepto gen, mediante las propiedades *idSymbol*, *posInicial*, *posFinal* y *longitud*, como la secuencia de referencia del gen mediante el atributo *secuencia*. Mientras que en el contexto del CSHG se tienen diferenciados los conceptos *Gene*, *Allele*, *Allelic Variant* y *Allelic Reference Type*, en el contexto de Diagen en la fase Conocimiento, únicamente es necesario hacer una referencia al gen del análisis, por eso, se

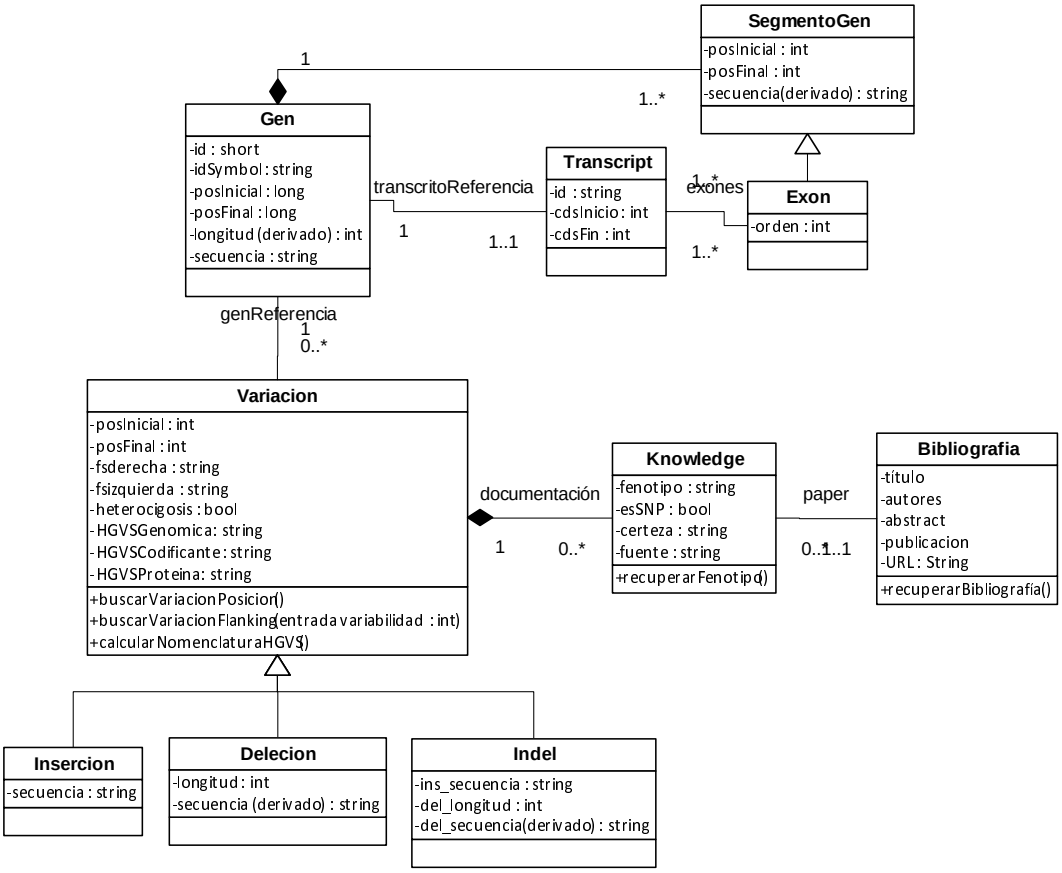


Figura 6.3: Modelo Conceptual Conocimiento

realizada una simplificación y agrupamiento dentro del concepto *Gen*. La entidad *SegmentoGen* es un subelemento del *Gen* con un significado determinado, por ejemplo, el elemento *Exon*, es un trozo del gen que se transcribe para la producción de proteínas. Un conjunto elementos *Exon* forman un elemento *Transcrito*, que es el conjunto de segmentos del gen que se transcriben.

El gen involucrado en el análisis sirve de referencia a las variaciones caracterizadas, modeladas por la entidad *Variacion*. Esta entidad representa cada una de las variaciones genéticas de las cuales se va a extraer el conocimiento. Una *Variacion* se define mediante los atributos *posInicial*, *posFinal*, que indican las posiciones inicial y final respecto de la secuencia de referencia del gen, los atributos *fsderecha* y *fsizquierda*, que indican las regiones adyacentes de la variación genética y el atributo *heterocigosis* que indica si la variación se ha producido en uno o en los dos alelos del gen del paciente. Además una variación se caracteriza mediante la nomenclatura HGVS, el estándar para la descripción de variaciones que consiste en *HGVSGenomica*, *HGVSCodificante* y *HGVSProteina*, que describe la variación respecto a los nucleótidos del gen, respecto a los aminoácidos codificantes o respecto a las proteínas respectivamente. Esta caracterización de *Variacion* se corresponde con la caracterización del CSHG excepto por la nomenclatura de HGVS y el atributo *heterocigosis*. En el contexto de Diagen, las variaciones representan alteraciones que se producen específicamente en los genes de los pacientes a analizar, por lo que una variación genética se debe caracterizar de manera que

contemple que el cambio se puede producir en uno o en los dos alelos.

Las variaciones genéticas pueden ser de tres tipos: *Insercion*, *Delecion* e *Indel*, y los atributos que los caracterizan se corresponden con el CSHG. Sin embargo, en este modelo no se especializa *Indel* como *SNP*, ya que esa información no responde a una característica intrínseca de la variación genética, sino que es una caracterización que un indel obtiene cuando se proporciona conocimiento sobre él.

Cuando las instancias de *Variacion* se buscan en el sistema de información genómico, se extrae conocimiento adicional acerca de ellas, información modelada mediante la entidad *Knowledge*. El conocimiento adquirido se define mediante los atributos: *phenotype*, que en el contexto de Diagen corresponde a la enfermedad que ha provocado la variación; *esSNP*, si la variación es un indel de un nucleótido que tiene un efecto no dañino y que además aparece en la población con una frecuencia al menos de un 1 %; *certeza*, que define el grado de certeza del conocimiento adquirido sobre la variación, y por último el atributo *fuente*, que indica de qué fuente de datos original se ha recopilado el conocimiento de esta información. En el CSHG está información se corresponde con las entidades *Certainty*, *Phenotype* y *SNP*, que representan conceptos que se asocian a las variaciones, pero que, en el contexto de Diagen, son consideradas en conjunto como información adicional de la variación.

El conocimiento adquirido sobre una variación está respaldado por una publicación científica, modelado mediante la entidad conceptual *Bibliography*. En el CSHG está información se corresponde con las entidades *Bibliography Reference* y *Bibliography DB*.

Para llevar a cabo la fase de conocimiento y proporcionar la funcionalidad expresada en los casos de uso es necesario proveer a las clases de métodos que ejecuten la funcionalidad necesaria.

Para llevar a cabo el caso de uso "Analizar Diferencias" es necesario crear una instancia de *Variacion* para cada una de las diferencias de la lista a analizar, inicializar los atributos correspondientes y ejecutar los métodos:

Caso uso "Búsqueda Variaciones" Para llevar a cabo este caso de uso se utilizan los dos métodos Buscar Variacion de la clase Variacion.

Caso uso "Buscar Variacion por Posición" El criterio de búsqueda es la posición

buscarVariacionPosicion()

Este método busca en el sistema de información genómico la variación mediante el criterio posición. Para ello se encarga de:

1. Realizar una consulta de todas las variaciones del mismo gen, tipo, cambio y posición.
2. Devuelve cierto si encuentra, al menos, una variación que cumpla los requisitos.

Caso uso "Buscar Variacion por Flanking Sequences" El criterio de búsqueda es por regiones adyacentes o flanking sequences

buscarVariacionFlanking(entrada porcentajeVariabilidad)

Este método se encarga de buscar en el sistema de información genómico la variación mediante el criterio flanking sequences. Para ello se encarga de:

1. Realizar una consulta de todas las variaciones del mismo gen, tipo, y cambio.
2. Para todas las variaciones recuperadas se comparan las flanking sequences y se establece el valor de variabilidad.
3. Se devuelve cierto si se encuentra al menos una variación cuyo porcentaje de Variabilidad sea menor que el porcentaje de Variabilidad de entrada.

Caso uso "Búsqueda Fenotipo y Bibliografía" Si la variación se ha encontrado en el sistema de información, se recupera la información asociada al fenotipo y a la bibliografía mediante los métodos:

recuperarFenotipo()

Para cada variación recuperada en la búsqueda de una variación se crea una instancia de Fenotipo y una instancia de Bibliografia. Este método se encarga de buscar en el sistema de información genómico la información relacionada con el fenotipo. Para ello se encarga de:

1. Realizar una consulta del fenotipo, certeza, caracterización de SNP y la fuente de datos original.
2. Crear una instancia de Bibliografía y recuperar la bibliografía asociada mediante la llamada al método:

   recuperarBibliografia()

   Para cada fenotipo recuperado la entidad Bibliografía recupera en el sistema de información genómico la información relacionada con la publicación asociada. Para ello se encarga de:

   a) Realizar una consulta de la información de publicación: titulo, autor, abstract, publicación, fecha y URL.

6.5. Implementación

El desarrollo de esta fase ha consistido en realizar la explotación de datos genómicos de dos sistemas de información genómica (Figura 6.4). En primer lugar, se ha integrado el acceso

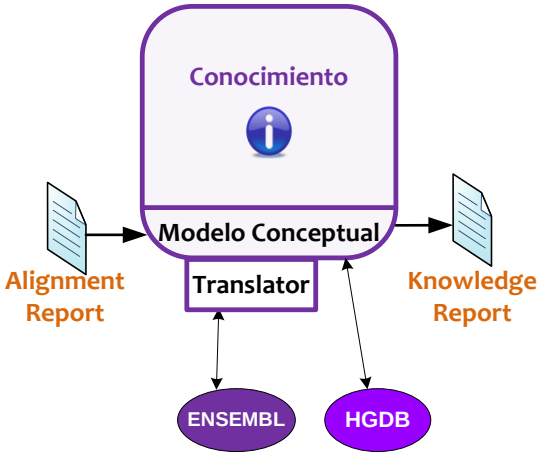


Figura 6.4: Fase Conocimiento

al sistema de información HGDB, creado en el proyecto Diagen. En segundo lugar, se ha integrado el acceso al sistema de información de ENSEMBL.

Para llevar a cabo la integración con ENSEMBL es necesaria la creación de un traductor para que sea capaz de realizar las consultas adecuadas al sistema de información y devolverlas en términos del modelo conceptual.

Para llevar a cabo la integración del sistema de información genómico HGDB no es necesario ningún traductor adicional, ya que el modelo conceptual de la fase de Conocimiento es una vista del modelo conceptual dLa HGDB. Para la interacción con esta base de datos se ha diseñado una capa de acceso a datos formada por los métodos siguientes:

- *RetrieveGeneById (idSymbol):* Obtiene un objeto de tipo Gen a partir de la información de la base de datos. Para ello recupera la información del gen consultándolo por nombre. El nombre (idSymbol) debe seguir el formato de HGNC.
- *RetrieveGeneByRefsource (refsource):* Obtiene un objeto de tipo Gen a partir de la información de la base de datos. Para ello recupera la información del gen consultándolo por identificador de la referencia. El identificador (refsource) debe seguir el formato de NCBI para identificar RefSeq.
- *CheckVariationByPos(var):* Busca la variación que recibe por argumento en la base de datos. Para ello busca por gen, por posición por tipo de variación y por el cambio producido.
- *CheckVariationByFlanking(var, sensibility)*: Busca la variación que recibe por argumento en la base de datos. Para ello busca por gen, por tipo de variación, por el cambio producido y además compara las regiones adyacentes para localizar la variación. Para las regiones adyacentes se establece un valor de variabilidad que determina si dos regiones son lo suficientemente similares para considerar que son la misma región adyacente.

- *RetrieveKnowledge()*: Recupera el efecto de la variación, mutante o snp, el fenotipo y la certeza de dicha información.
- *RetrieveBibliography():* Recupera los datos de la referencia bibliográfica que garantiza la relación entre la variación y el fenotipo, es decir, titulo, autor, abstract, publicación y url.

6.5.1. Modelo Conceptual Reporte de Resultados

Se diseña un modelo conceptual de Reporte de Resultados, *VariationKnowledge* para expresar los resultados de manera precisa o, en caso de utilizarse en el contexto de otra herramienta, para guiar el proceso de flujo de datos.

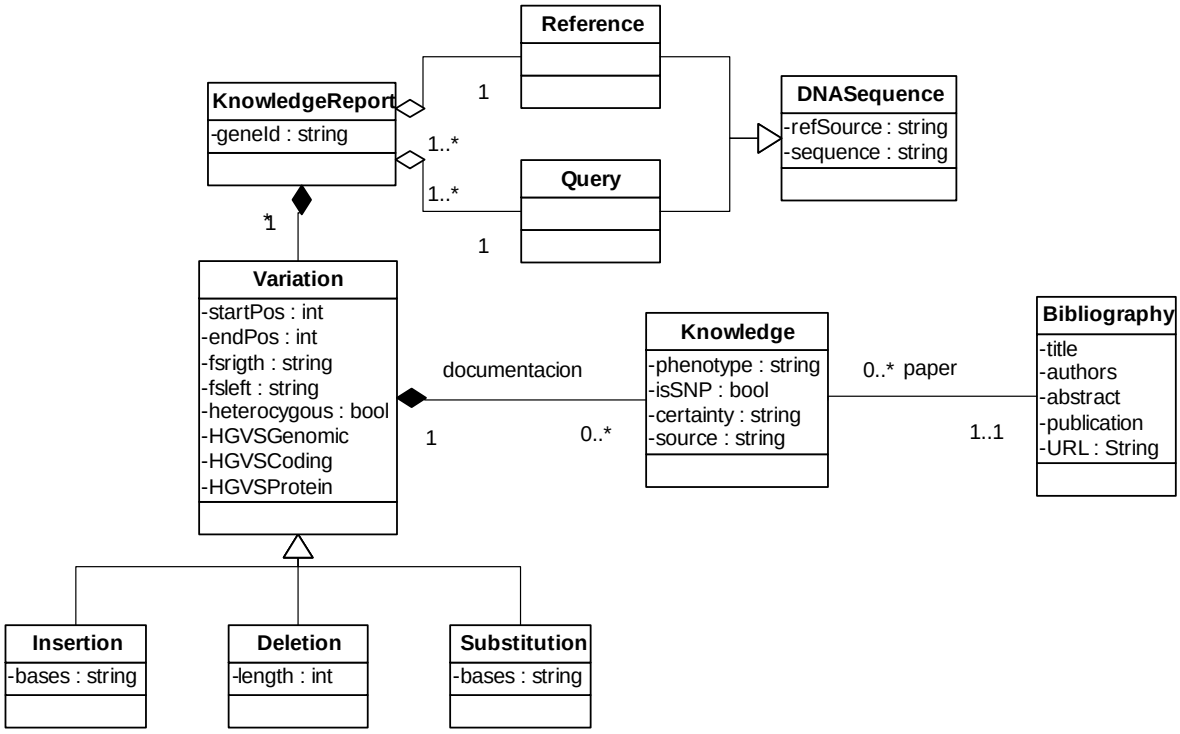


Figura 6.5: Modelo Conceptual KnowledgeReport

El modelo conceptual diseñado, KnowledgeReport, representa las variaciones genéticas y su caracterización fenotípica encontradas en una secuencia genética respecto a una secuencia genética de Referencia (Figura 6.5). Este modelo es una vista del modelo conceptual de la fase de Conocimiento. La entidad conceptual *KnowledgeReport* representa el reporte generado después de realizar la caracterización de las variaciones mediante la extracción de conocimiento del sistema de información genético, y el atributo *geneId* contiene el identificador del gen del reporte generado. En este modelo conceptual se incluyen los dos elementos utilizados para el análisis de variaciones, es decir, las secuencias genéticas *DNASequence*, *Reference* y *Query*. Como resultado de la comparativa y el análisis se ofrece la lista de variaciones *Variation* y en caso de haber sido localizada, su documentación *Knowledge* y el paper asociado *Bibliography*.

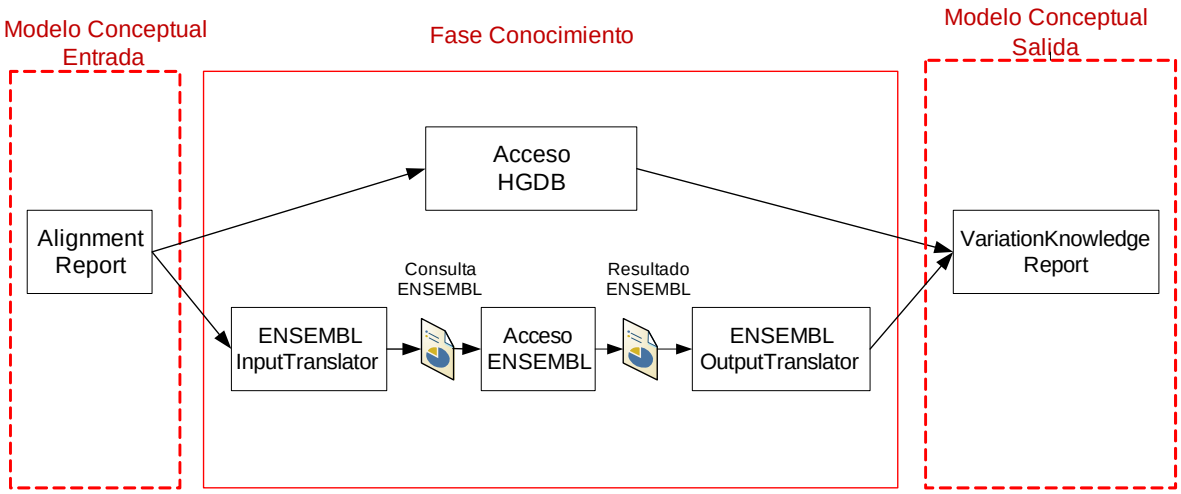


Figura 6.6: Integración Conocimiento implementado

6.5.2. Implementación del Modelo Conceptual Reporte de Resultados

La implementación de la fase de Conocimiento se ha desarrollado en Java y el modelo conceptual, de la misma manera que el SampleTreatmentReport y el AlignmentReport, se ha implementado mediante clases Java y mediante la especificación XMLSchema.

Una vez implementada la funcionalidad y los artefactos necesarios para expresar los resultados en términos del modelo conceptual, la fase de Conocimiento puede ejecutarse de manera independiente. En la Figura 6.7 se puede observar un ejemplo de la salida XML que proporciona la fase de Conocimiento ejecutada de manera independiente. Para ello, se ha utilizado una lista de diferencias del Gen BRCA2 donde ser han localizado dos indels respecto a la secuencia de referencia NG_012772.1. En la fase de conocimiento se ha obtenido que el primer indel es un SNP (rs3752451) localizado en la base de datos de dbSNP, y el segundo indel está asociado con el cáncer de mama y el paper que lo corrobora se titula “A common variant in BRCA2 is associated with both breast cancer risk and prenatal viability”, indicando su url en Pubmed. Además este indel se ha localizado también en dbSNP como un SNP (rs144848).

```
<KnowledgeReport geneId="BRCA2">
    <Reference id="Reference_Query_1">
        <refDB>NG_012772.1</refDB>
     </Reference>
    <Query id="Consensus_Query_1">
        <refFile>128-1.TXT</refFile>
    </Query>
    <Variations>
        <InDel  endPos="28439" initialPos="28439"
            fsl="AAGATCAAAGAACCTACTCT" fsr="TTGGGTTTTCATACAGCTAG"
            heterozygosis="false">
            <Change source="T">A</Change>
            <Documentations>
                <Documentation source="dbSNP">
                    <Phenotype certainty="0">SNP</Phenotype>
                    <Effect>SNP</Effect>
                    <Bibliography>
                        <title>rs3752451</title>
                    </Bibliography>
                </Documentation>
            </Documentations>
        </InDel>
        <InDel endPos="22113" startPos="22113"
            fsl="TTGTCCTGTTTAAAGCCATC" fsr="AGTTACAATAGATGGAACTT"
            heterozygous="true">
            <Change source="A">C</Change>
            <Documentations>
                <Documentation source="HGMD">
                    <Phenotype certainty="1">Breast cancer, association with</Phenotype>
                    <Effect>Mutant</Effect>
                    <Bibliography>
                        <title> A common variant in BRCA2 is associated with both breast cancer
                            and prenatal viability. </title>
                        <publication> CRC Department of Oncology, University of Cambridge,
                            Strangeways Research Laboratory, Cambridge, UK.
                            katie.healey@srl.cam.ac.uk </publication>
                        <date>2000-11-01+01:00</date>
                        <authors> Healey CS, Dunning AM, Teare MD, Chase D, Parker L, Burn J,
                            Chang-Claude J, Mannermaa A, Kataja V, Huntsman DG, Pharoah PD,
                            Luben RN, Easton DF, Ponder BA. </authors>
                        <url>http://www.ncbi.nlm.nih.gov/sites/entrez?cmd=Retrieve&amp;
                            db=PubMed&amp;list_uids=17148771&amp;dopt=Abstract</url>
                    </Bibliography>
                </Documentation>
                <Documentation source="dbSNP">
                    <Phenotype certainty="0">SNP</Phenotype>
                    <Effect>SNP</Effect>
                    <Bibliography>
                        <title>rs144848</title>
                    </Bibliography>
                </Documentation>
            </Documentations>
        </InDel>

    </Variations>
</KnowledgeReport>
```

Figura 6.7: Ejemplo XML VariationKnowledgeReport

Capítulo 7

Conclusiones, Publicaciones y Trabajo Futuro

En esta tesis de máster se ha modelado e implementado un framework, llamado Diagen, para la integración y explotación de información genómica a fin de soportar el análisis personalizado de secuencias de ADN. El objetivo del framework es proporcionar un marco formal para integrar la funcionalidad necesaria para la medicina genética personalizada, de forma que sirva para el desarrollo de aplicaciones bioinformáticas sistemáticamente.

Mediante la utilización del framework Diagen es posible el análisis de una muestra de ADN de un paciente para obtener la lista de variaciones y sus fenotipos asociados, que determinan si el paciente es potencialmente propenso a padecer una enfermedad. Para ello se han integrado todas las fases identificadas: Fase Tratamiento, Fase Alineamiento y Fase Conocimiento.

Actualmente, existen muchas herramientas de análisis de ADN funcionales que ofrecen resultados parciales a los investigadores en genética. Sin embargo, hasta el momento no existía una herramienta que fuera capaz de cubrir todo el proceso de análisis de secuencias genéticas de manera unificada e integrar, en una sola herramienta, toda la funcionalidad requerida de manera transparente para el usuario. El framework desarrollado soporta todo este proceso descargando a los usuarios de llevar a cabo manualmente la difícil tarea de conseguir el flujo de información entre herramientas. También sirve como un soporte inicial para futuros desarrollos de herramientas, dado su caracter modular.

Para realizar la identificación de las fases que cubre el framework, se ha desarrollado una descripción del proceso de negocio llevado a cabo por el Instituto de Medicina Genómica (IMeGen), empresa dedicada al análisis de secuencias de ADN para la elaboración de diagnósticos. Cada una de las fases del proceso se ha formalizado mediante la elaboración de un modelo conceptual de desarrollo. Para llevar a cabo esta formalización, se ha tenido en cuenta los aspectos biólogicos plasmados en el modelo conceptual CSHG que representa los conceptos biólogicos del genoma humano. Así mismo, el flujo de información entre fases está guiado por una vista de el modelo conceptual de cada una de las fases, al cual se ha denominado Reporte de Resultados.

Cada una de las fases implementadas integra un conjunto de herramientas que llevan a cabo las tareas necesarias. Para aquellas herramientas implementadas fuera del contexto del proyecto Diagen ha sido necesaria la implementación de un traductor. En cambio, para la nueva funcionalidad implementada en base al modelo conceptual de cada fase, la creación de este traductor no es necesaria.

Durante el análisis del proceso de negocio de IMeGen, se han observado un conjunto de problemas relacionados con las máquinas secuenciadoras, los algoritmos de alineamiento y la heterogeneidad de los sistemas de información genómicos. La nueva funcionalidad implementada tiene como objetivo dar solución a los problemas planteados que no han sido solucionados por las herramientas de análisis de ADN actuales.

La contribución del trabajo realizado es aplicar los principios del modelado conceptual en un entorno real, donde estos principios son generalmente obviados, como es el domino de la biología genómica. En consecuencia, este trabajo valida que el modelado conceptual es útil para entender el dominio y para desarrollar software que explote, analice e interprete información genómica. El desarrollo del framework Diagen proporciona resultados visibles para esta comunidad, ya que ayuda a los investigadores en su trabajo diario a la hora de realizar un análisis de secuencias genéticas de ADN. Por último destacar, que el valor del framework Diagen no solo es fruto del trabajo realizado en esta tesis de máster sino que está basado en un conjunto de trabajos previos en el contexto del proyecto como son: la especificación de un modelo conceptual del genoma humano, la carga de datos de bases de datos genómicas relevantes y una primera aproximación a la explotación de datos genómicos para obtener la problemática a abordar.

7.1. Publicaciones

Los resultados de esta tesis de máster se han publicado la conferencia internacional llamada Bioinformatics, que forma parte de BIOSTEC (International Joint Conference on Biomedical Engineering Systems and Technologies) celebrada en Roma (Italia) en Enero de 2011.

Villanueva M. J., Valverde F. and Pastor O.: Applying Conceptual Modeling To Alignment Tools: One Step Towards the Automation of DNA Sequence Analisis. International Conference on Bioinformatics Models, Methods and Algorithms (BIOINFORMATICS 2011), 2011, Rome (Italy)

Además, se ha participado en varias publicaciones relacionadas, dentro del proyecto Diagen desarrollado en el Centro Pros durante los últimos dos años.

Pastor, O.; Levin, A. M.; Celma, M.; Casamayor, J. C.; Eraso, L. E.; Villanueva, M. J. & Perez-Alonso, M. Enforcing Conceptual Modeling to Improve the Understanding of Human Genome RCIS, 2010, 85-92

Martinez AM, Martin A., Villanueva M. J., Valverde F, Levin AM and Pastor O.: Facing the Challenges of Genome Information Systems: a Variation Analysis Prototype. Caise Forum 2010 (Poster). Hammammet (Tunisia)

Martinez AM, Martin A., Villanueva M.J., Valverde F, Levin AM and Pastor O.: Facing the Challenges of Genome Information Systems: a Variation Analysis Prototype. Lecture Notes in Business Information Processing, 2011, Volume 72, 222-237

7.2. Trabajo Futuro

El framework Diagen es una primera toma de contacto con el problema general a abordar. Como trabajo futuro, la primera linea de actuación es continuar con el desarrollo del framework. En concreto se tienen que abordar los siguientes puntos:

- Finalizar correctamente la implementación de todas las fases planteadas: Por ejemplo la implementación de todos los casos de uso de la fase de secuenciación para ofrecer un servicio completo en el contexto del framework Diagen. También debe completarse la fase de reporte final que actúe de interfaz con el usuario final, en este caso un médico o biólogo.
- Añadir nueva funcionalidad bioinformática enfocada al análisis genético: la creación de primers para secuenciación, la ampliación de funcionalidad a la hora de analizar diferencias, la relación de las variaciones con las rutas metabólicas (pathways) involucradas o también el análisis del cambio en base a la estructura de las nuevas proteínas. Esta nueva funcionalidad biológica implicará analizar previamente el impacto en el modelo conceptual involucrado.
- Elaboración de un entorno de desarrollo basado en el paradigma orientado a servicios para la creación de herramientas bioinformáticas personalizadas. Utilizando el framework Diagen como base, se definirán servicios especializados e integrables que aborden las tareas bioinformáticas planteadas. Estos servicios proporcionarán la misma funcionalidad que el framework, pero sin la necesidad de conocer las peculiaridades tecnológicas del mismo. Mediante la interconexión de estos servicios con otros existentes, será posible la composición de herramientas más complejas.

Como segunda linea de actuación se pretende completar el soporte al diagnóstico mediante el desarrollo de funcionalidad adicional que de el soporte al tratamiento. De esta manera se conseguirá abordar el reto de la medicina personalizada de determinar qué fármaco es el más adecuado para un paciente concreto. En concreto las tareas futuras a abordar son:

- Ampliación del modelo conceptual del genoma humano, enfocada a la medicina, para la incorporación de fármacos y tratamientos junto con las relaciones con la información genética.
- Añadir nueva funcionalidad bioinformática para el diagnóstico de tratamientos adecuados basados en análisis genéticos. Esta funcionalidad utilizara el nuevo modelo conceptual planteado.
- Incorporar a la herramienta desarrollada, la identificación del tratamiento más adecuado para el individuo asociado a su configuración genotípica. Como tarea previa será necesaria la creación de una base de conocimiento, que permita la toma de decisiones.

Bibliografía

[1] Ronald Adkins. Comparison of the accuracy of methods of computational haplotype inference using a large empirical dataset. *BMC Genetics*, 5(1):22, 2004.

[2] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403 – 410, 1990.

[3] Mark Anderson. The road to the personal genome. *IEEE Spectr.*, 47:68–68, February 2010.

[4] Khalid Awadelkarim et al. Brca1 and brca2 status in a central sudanese series of breast cancer patients: interactions with genetic, ethnic and reproductive factors. *Breast Cancer Research and Treatment*, 102:189–199, 2007. 10.1007/s10549-006-9303-z.

[5] Danielson PB Azadan RJ, Fogleman JC. Capillary electrophoresis sequencing: maximum read length at minimal cost. *Biotechniques*, 32:24–6, Jan 2002.

[6] Tushar R Bhangale, Matthew Stephens, and Deborah A Nickerson. Automating resequencing-based detection of insertion-deletion polymorphisms. *Nat Genet*, 38(12):1457–1462, December 2006.

[7] Applied Biosystems. Seqscape. http://www.apliedbiosystems.com.

[8] Cecile Brachet, Julia Birk, Catherine Christophe, Sylvie Tenoutasse, Brigitte Velkeniers, Claudine Heinrichs, and Jonas Rutishauser. Growth retardation in untreated autosomal dominant familial neurohypophyseal diabetes insipidus caused by one recurring and two novel mutations in the vasopressin-neurophysin ii gene. *Eur J Endocrinol*, pages EJE–10–0823, 2010.

[9] P Curtis C Bromberg, H Cash and CJ Goebel. *Sequencher, Gene Codes Corporation.* Ann Arbor, Michigan, 1995.

[10] Peter J. A. Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel J. L. de Hoon. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 2009.

[11] Codon Code Corporation. Codon code aligner. http://www.codoncode.com/aligner.

[12] Johan T. den Dunnen and Stylianos E Antonarakis. Mutation nomenclature extensions and suggestions to describe complex mutations: A discussion. *Human Mutation*, 15(1):7–12, 2000.

[13] Dmitry A. Dmitriev and Roman A. Rakitov. Decoding of superimposed traces produced by direct sequencing of heterozygous indels. *PLoS Comput Biol*, 4(7):e1000113, 07 2008.

[14] Andreas Doring, David Weese, Tobias Rausch, and Knut Reinert. Seqan an efficient, generic c++ library for sequence analysis. *BMC Bioinformatics*, 9(1):11, 2008.

[15] Kevin Garwood, Christopher Garwood, Cornelia Hedeler, Tony Griffiths, Neil Swainston, Stephen Oliver, and Norman Paton. Model-driven user interfaces for bioinformatics data resources: regenerating the wheel as an alternative to reinventing it. *BMC Bioinformatics*, 7(1):532, 2006.

[16] Victor Gold. *Compendium of chemical terminology: IUPAC recommendations*. ISBN 0632017651. Blackwell Scientific Publications, 1987.

[17] Margaret A. Hamburg and Francis S. Collins. The path to personalized medicine. *New England Journal of Medicine*, 363(4):301–304, July 2010.

[18] R. C. G. Holland, T. A. Down, M. Pocock, A. Prlic, D. Huen, K. James, S. Foisy, A. Dräger, A. Yates, M. Heuer, and M. J. Schreiber. BioJava: an open-source framework for bioinformatics. *Bioinformatics*, 24(18):2096–2097, 2008.

[19] Alberto Labarga, Franck Valentin, Mikael Anderson, and Rodrigo Lopez. Web Services at the European Bioinformatics Institute. *Nucleic Acids Research*, 35(suppl 2):W6–W11, 2007.

[20] Ming-Jen Lee et al. Identification of forty-five novel and twenty-three known nf1 mutations in chinese patients with neurofibromatosis type 1. *Hum. Mutat.*, 27(8):832–832, 2006.

[21] Carl Manaster, Weiyue Zheng, Markus Teuber, Stefan Wachter, Frank Daring, Stefan Schreiber, and Jochen Hampe. InSNP: A tool for automated detection and visualization of SNPs and InDels. *Hum. Mutat.*, 26(1):11–19, 2005.

[22] E. Ort and B. Mehta. Java architecture for xml binding (jaxb). Technical Report Sun Developer Network, March 2003.

[23] Oscar Pastor. Conceptual modeling meets the human genome. In Qing Li, Stefano Spaccapietra, Eric Yu, and Antoni Olivé, editors, *Conceptual Modeling - ER 2008*, volume 5231 of *Lecture Notes in Computer Science*, pages 1–11. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-87877-3-1.

[24] John MC Ionides Wim F Vranken Tim J Stevens Rasmus H Fogh, Wayne Boucher and Ernest D Laue. Memops: Data modelling and automatic code generation. *Journal of Integrative Bioinformatics*, 7, 2010.

[25] Nicole Rusk. Focus on next-generation sequencing data analysis. *Nature Methods*, 6(11s):S1, October 2009.

[26] Paul Rutherford, Walt Abell, Clare Churcher, Alan McKinnon, and John McCallum. Usability of navigation tools for browsing genetic sequences. In *Proceedings of the Eleventh Australasian Conference on User Interface - Volume 106*, AUIC '10, pages 33–41, Darlinghurst, Australia, Australia, 2010. Australian Computer Society, Inc.

[27] Jeoffrey Schageman, Christopher Horton, Sijing Niu, Harold Garner, and Alexander Pertsemlidis. Elxr: a resource for rapid exon-directed sequence analysis. *Genome Biology*, 5(5):R36, 2004.

[28] Jay Shendure and Hanlee Ji. Next-generation dna sequencing. *Nat Biotech*, 26(10):1135–1145, October 2008.

[29] Softgenetics. Mutation surveyor. http://www.softgenetics.com.

[30] Ola Spjuth, Tobias Helmus, Egon Willighagen, Stefan Kuhn, Martin Eklund, Johannes Wagener, Peter Murray-Rust, Christoph Steinbeck, and Jarl Wikberg. Bioclipse: an open source workbench for chemo- and bioinformatics. *BMC Bioinformatics*, 8(1):59, 2007.

[31] Jason E. Stajich, David Block, Kris Boulez, Steven E. Brenner, Stephen A. Chervitz, Chris Dagdigian, Georg Fuellen, James G.R. Gilbert, Ian Korf, Hilmar Lapp, Heikki Lehväslaiho, Chad Matsalla, Chris J. Mungall, Brian I. Osborne, Matthew R. Pocock, Peter Schattner, Martin Senger, Lincoln D. Stein, Elia Stupka, Mark D. Wilkinson, and Ewan Birney. The Bioperl Toolkit: Perl Modules for the Life Sciences. *Genome Research*, 12(10):1611–1618, 2002.

[32] Elspeth Bruford Mathew Wright Michael Lush Sue Povey, Ruth Lovering and Hester Wain. The hugo gene nomenclature committee (hgnc). *Human Genetics*, 109:678–680, 2001.

[33] Kevin Thornton. libsequence: a C++ class library for evolutionary genetic analysis. *Bioinformatics*, 19(17):2325–2327, 2003.

[34] Siyan K. Vakatov D. and Ostell J., editors. *The NCBI C++ Toolkit*. National Center Biotechnology Information, 2003.

[35] J. Venter et al. The sequence of the human genome. *Science*, 291(5507):1304–1351, 2001.

[36] Michael S. Waterman, Temple F. Smith, and H. L. Katcher. Algorithms for restriction map comparisons. *Nucleic Acids Research*, 12(1):237–242, 1984.

Apéndice A

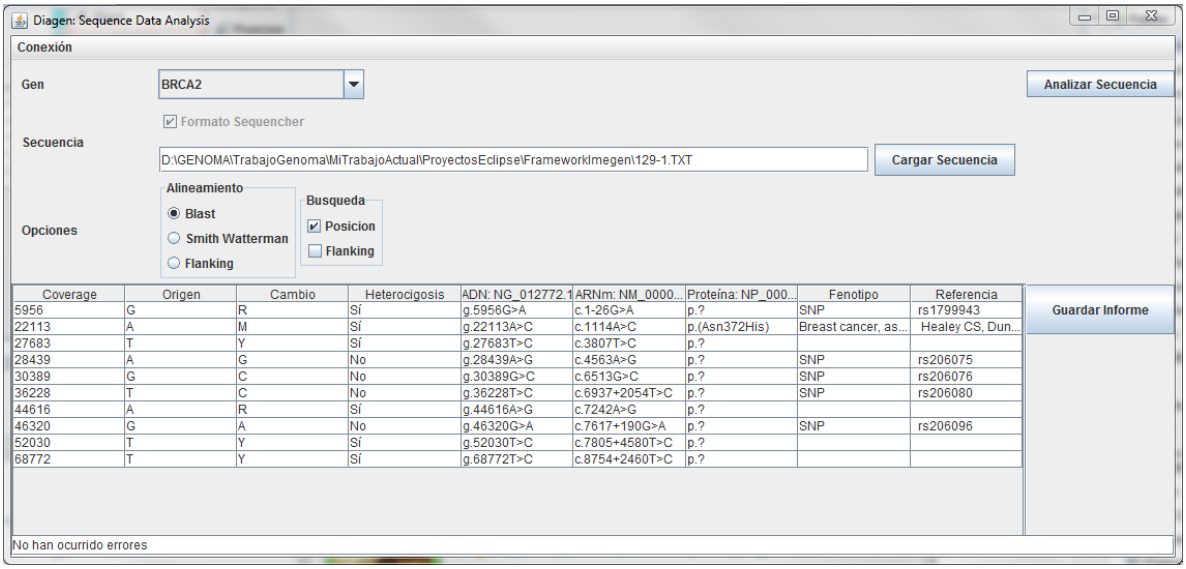
Glosario

- ADN (Ácido Desoxirribonucleico): Macromolécula que contiene el material genético de una célula. Es la responsable de la creación de proteínas responsables del desarrollo y funcionamiento de seres vivos, así como de la transmisión hereditaria.
- Cromosoma: Cada una de las estructuras en las que se divide el ADN en cada célula. El ser humano está formado por 23 pares de cromosomas homólogos, donde cada uno de ellos proviene de cada progenitor (uno de la madre y otro del padre).
- Nucleótidos: Moléculas, que unidas entre si, forman la estructura del ADN. Cada nucleótido contiene una base, que puede ser de cuatro tipos: adenina (A), guanina (G), citosina (C) o timina (T).
- Genoma: Conjunto de todo el material genético contenido en cada cromosoma. Está formado por la sucesión de todos los nucleótidos que forman el ADN de una célula. El genoma humano contiene aproximadamente 3.000 millones de pares de nucleótidos.
- Gen: Subconjunto de nucleótidos del genoma responsables de la creación de proteínas para el desarrollo o funcionamiento de una función fisiológica específica.
- Secuencia genómica: Secuencia lineal de todos los nucleótidos del genoma.
- Secuencia genética: Secuencia lineal de todos los nucleótidos de un gen.
- Genotipo: Contenido del genoma de un individuo concreto.
- Fenotipo: Conjunto de características externas que se manifiestan en un individuo concreto a causa de su genotipo y el ambiente.
- Secuenciación: Métodos y técnicas bioquímicas utilizadas para obtener el genotipo de un individuo a partir de una muestra de ADN.
- Electroferograma: Gráfico de representación del ADN. Está formado por un conjunto de señales que representan la sucesión de nucleotidos que forman el ADN.

- Secuencia de referencia: Sucesión de nucleótidos ficticia que pretende representar la secuencia de nucleótidos original de todos los individuos de una especie.
- Variación: Cambio en un nucleótido de una secuencia genética o genómica de un individuo respecto a la secuencia de referencia de su especie.
- Mutación: Variación que tiene un efecto en la creación de proteínas y tiene como consecuencia la aparición de una enfermedad genética.
- Alelo: Cada uno de los valores de los nucleótidos de cada par de cromosomas del ADN que situados en la misma posición dentro del genotipo.
- Variación homocigota o en homocigosis: Variación que se produce en los dos alelos del genotipo, es decir, que se produce en los dos cromosomas.
- Variación heterocigota o en heterocigosis: Variación que solo se produce en un alelo del genotipo, es decir, que se produce en un solo cromosoma.

Apéndice B

Interfaz Imegen



Coverage	Origen	Cambio	Heterocigosis	ADN: NG_012772.1	ARNm: NM_0000...	Proteína: NP_000...	Fenotipo	Referencia
5956	G	R	Sí	g.5956G>A	c.1-26G>A	p.?	SNP	rs1799943
22113	A	M	Sí	g.22113A>C	c.1114A>C	p.(Asn372His)	Breast cancer, as...	Healey CS, Dun...
27683	T	Y	Sí	g.27683T>C	c.3807T>C	p.?		
28439	A	G	No	g.28439A>G	c.4563A>G	p.?	SNP	rs206075
30389	G	C	No	g.30389G>C	c.6513G>C	p.?	SNP	rs206076
36228	T	C	No	g.36228T>C	c.6937+2054T>C	p.?	SNP	rs206080
44616	A	R	Sí	g.44616A>G	c.7242A>G	p.?		
46320	G	A	No	g.46320G>A	c.7617+190G>A	p.?	SNP	rs206096
52030	T	Y	Sí	g.52030T>C	c.7805+4580T>C	p.?		
68772	T	Y	Sí	g.68772T>C	c.8754+2460T>C	p.?		

Figura B.1: Interfaz Imegen

En la Figura B.1 se puede ver la interfaz utilizada para el análisis de una muestra de ADN. Mediante la pestaña *Gen*, se puede seleccionar el gen del análisis. En la casilla *Secuencia* se selecciona el fichero que contiene la información de la muestra genética. En las casillas de *Opciones* se selecciona, por un lado, la herramienta de alineamiento que se desea utilizar (Blast, SW o Flanking), y por otro lado, la estrategia de búsqueda en el sistema de información genómico (Posición o Flanking).

Se debe tener en cuenta, que como la herramienta sequencher es una herramienta comercial, no es posible modificar su implementación, por lo que de momento, se exportan los resultados a un fichero y se importan en la herramienta.

Una vez seleccionado toda la información necesaria se hace click en el botón Analizar Secuencia para llevar a cabo el análisis. Una vez finalizado, se muestra una tabla con los resultados: Una

fila por cada variación localizada y una columna indicando la información necesaria para describirla. Una vez finalizado el análisis, la tabla de resultados se puede exportar en un formato soportado por Excel.

B.1. Codigo Fuente

org.pros.genoma.diagen.userInterface

Incluye clases Imegen.java y Analysis.java.

Analysis.java

```
public class Analysis {
        private String idGene;
        private String filePath;
        private Vector<Vector> dataTable;
        private Vector<String> header;

        private GenomaDb genomaDb;

        public Analysis(){
        }

        /*--------Public Methods: Analysis-----------------------------*/
        /**
         * Executes all Analysis Phases
         *@param application Application used to perform the SampleTreatmentPhase
         *@param alignmentAlgorithm Algorithm to perform the AlignmentPhase
         *@param searchStrategy Strategy to perform the VariationKnowdlegePhase
         */
        public DefaultTableModel runAnalysis(SampleTreatment.Application application,
            Alignment.Algorithm alignmentAlgorithm,Strategy searchStrategy){ //Create
            report
                /*PERFORM ANALYSIS*/
                /*File Names*/
                String fileNameSampleTreatmentXML="SampleTreatmentReport"+this.idGene+
                    ".xml";
                String fileNameAlignment="AlignmentAlgReport"+this.idGene+".xml";
                String fileNameVariationKnowdlegeXML= "VariationKnowdlegeReport"+this.
                    idGene+".xml";


                /*1. SAMPLE TREATMENT PHASE*/
                SampleTreatment sampleTreatmentPhase=new SampleTreatment(this.idGene,
                    application);
                sampleTreatmentPhase.executeSampleTreatment(this.filePath);
                SampleTreatmentReport stReport=sampleTreatmentPhase.
                    performResultTransformation();
                stReport.writeSampleTreatmentReportToFile(fileNameSampleTreatmentXML);

                /*2. ALIGNMENTPHASE*/
                Alignment alignmentPhase=new Alignment(this.idGene, alignmentAlgorithm
                    ,this.genomaDb);
                alignmentPhase.performInputTransformation(stReport);
                alignmentPhase.executeAlignment();
                AlignmentReport aReport=alignmentPhase.performResultTransformation();
```

```
                aReport.writeAlignmentReportToFile(fileNameAlignment);

                /*3. VARIATION KNOWLEDGE PHASE*/
                Knowledge variationKnowledgePhase= new Knowledge(this.idGene,
                    searchStrategy,this.genomaDb);
                variationKnowledgePhase.performInputTransformation(aReport);
                variationKnowledgePhase.executeVariationKnowledge();
                VariationKnowledgeReport vkReport=variationKnowledgePhase.
                    performResultTransformation();
                vkReport.writeVariationKnowdlegeReportToFile(
                    fileNameVariationKnowdlegeXML);


                /*4. Fill report data table*/
                /*Report.jar code*/
                /*Process VariationKnowledgeReport data and convert it into the
                    required formats*/
                org.pros.genoma.report.GenomaDb genomaDbAna = new org.pros.genoma.
                    report.GenomaDb ("*****");
                ReportC rep = new ReportC();
                rep.retrieveReport(fileNameVariationKnowdlegeXML);
                Reporter reporter = new Reporter(rep, genomaDbAna);
                Vector data= reporter.createTable();

                /*Visualization of the report in a Table*/
                DefaultTableModel dtm=this.fillReportData(data);

        return dtm;
        }

/**
 * Fills the dataTable from a Vector that contains the header in the first element
 * and the rows in the next elements.
 * @param data The vector containing the data
 * @return A DefaultTableModel object containing the columns and the rows.
 */
        private DefaultTableModel fillReportData(Vector<Vector> data){
                DefaultTableModel dtm=new DefaultTableModel();
                this.dataTable=data;
                //First element: Header
                this.header=data.get(0);
                data.remove(0);
                for(String s:header){
                        dtm.addColumn(s);
                }
                //Next elements: Rows
                for(Vector v:data){
                        dtm.addRow(v);
                }
                return dtm;
        }

        /**
         * Copies the data from the dataTable in a Excel readable format
         * @param outputFileName Name of the File to save the data
         */
        public void saveToExcel(String outputFileName) {
                try {
                        PrintWriter fileOut;
                        fileOut = new PrintWriter(new FileWriter(outputFileName));
                        /*Print cabecera*/
                        fileOut.print(this.header.get(0));
                        for (String s: this.header) {
                                fileOut.print(s+";");
                        }
```

```
                    fileOut.println();
                    /*Print content*/
                for (int i = 0; i < this.dataTable.size(); i++) {
                    fileOut.print(this.dataTable.get(i).get(0));
                    for (int j = 1; j < this.dataTable.get(i).size(); j++) {
                        Object object = this.dataTable.get(i).get(j);
                        if (object != null)
                          fileOut.print(";" + object.toString());
                        else
                          fileOut.print(";");
                    }
                    fileOut.println();
                 }
                 fileOut.close();
                } catch (IOException e) {
                        e.printStackTrace();
                        }
        }
}
```

Apéndice C

Implementación Framework Diagen

C.1. Fase Tratamiento

org.pros.genoma.diagen.treatment

Incluye las clases ISampleTreatmentOperations.java, SampleTreatment.java, SampleTreatmentReport.java, SampleTreatmentResult.java, SequencherOutput.java y SequencherOutputTranslator.java.

SampleTreatment.java

```
/**
 * Class that executes the SampleTreatment Phase:
 * 1. Activities to execute a sample treatment: Exons Assembly and Nucleotide curation
 * 2. The output of these activities is saved in a file.
 * 3. A SampleTreatmentResult is created
 * 4. The output data saved in a file is imported in the SampleTreatmentResult
 * 5. A SampleTreatmentReport is created (to exchange data with the next phase)
 * 6. The SampleTreatmentResult is translated to a SampleTreatmentReport
 *
 * Steps 1-4 are performed by the public method executeSampleTreatment()
 * (3-4 by the private method retrieveSampleTreatmentResult())
 * Steps 5-6 are performed by the public method performResultTransformation()
 *
 *
 * @author mariajo
 *
 */
public class SampleTreatment {
        static public enum Application{Sequencher};
        String geneId;
        Application application;
        SampleTreatmentResult result;
        String SampleTreatmentOutputFileName;

        /**
         * @param geneId Gene Identifier of the analysis
```

```
 * @param application Application/method used to performed the sample
     treatment
 */
public SampleTreatment(String geneId, Application application) {
        super();
        this.geneId = geneId;
        this.application=application;
}

/**
 * Executes the activities required to perform the Sample Treatment and
 * creates a SampleTreatmentResult filled with the output of those activities.
 *
 * @param outputFileName Name of the file to save the sample treatment output
 *
 * */
public void executeSampleTreatment(String outputFileName){
        /*1. Code to Execute Sample Treatment*/
        /* As the sample treatment is performed outside the application
            environment,
          * it does not perform any activity. The results are already loaded on
              the outputFileName*/
        this.SampleTreatmentOutputFileName=outputFileName;
        /*End Code to Execute Sample Treatment*/

        /*2. Result operations: Creates and fills a Result with the output of
            the
          * previous Sample Treatment*/
        this.retrieveSampleTreatmentResult();
}

/**
 * Translates the SampleTreatmentResult into a SampleTreatmentReport
 *
 * @return A SampleTreatmentReport containing sample treatment data
 */
public SampleTreatmentReport performResultTransformation(){
        SampleTreatmentReport report=null;
        ResultTranslator translator=null;
        /*Choose application: Create the corresponding translator*/
        if(this.application.compareTo(Application.Sequencher)==0){
                translator=new SequencherResultTranslator((SequencherResult)
                    this.result);
        }/*else if() for other applications*/

        /*Transformation: Execute the transformation*/
        report=(SampleTreatmentReport)translator.resultToReport();
        return report;
}

/*-------------------Private Methods-----------------------------------*/
/**
 * Creates and fills a SampleTreatmentResult with the output of the Sample
     Treatment
 * execution.
 *
 * */
private void retrieveSampleTreatmentResult(){
        /*Choose application: Create the corresponding Result*/
        if(this.application.compareTo(Application.Sequencher)==0){
                this.result=new SequencherResult(this.geneId);
                /*Result retrieval: load data from file*/
                ((SequencherResult)this.result).retrieveResultFromFile(this.
                    SampleTreatmentOutputFileName);
        }/*else if() for other applications*/
```

```
        }
}
```

SampleTreatmentReport.java

```
/**
 * SampleTreatmentReport is a class that contains the implementation of
 * the conceptual model SampleTreatmentReport.
 * Actually it contains an attribute pointing the root object to the JAVA Objects
 * collection of the implementation.
 * The implementation of the conceptual model is performed with XMLSchema. In order
 * to work with XML and Java it has been used the technology JAXB.
 *
 * This class has five methods to create, set and get the properties of
 * these objects using this technology:
 * 1. Set the identifier of the gene of the SampleTreatment data
 * 2. Get the identifier of the gene of the SampleTreatment data
 * 3. Set the sample sequence (query) and their decomposition in several sequenced
     segments (exons)
 * 4. Get the sample sequence (query).
 * 5. Get the sequenced segments (exons) of the sample sequence (query).
 *
 * @author mariajo
 *
 */
public class SampleTreatmentReport extends Report implements
    ISampleTreatmentReportOperations{
        private SampleTreatmentReportCM sampleCM;


        public SampleTreatmentReport(){
                this.sampleCM=new SampleTreatmentReportCM();
        }

        /**
         * Sets the value of the attribute idGene in the SampleTreatmentReport object.
         *
         * @param geneId Identifier of the gene of the sample
         */
        public void setGeneId(String geneId){
                /*Set idGene in the SampleTreatmentReportCM*/
                this.sampleCM.setIdGene(geneId);
        }
        /**
         * Gets the value of the attribute idGene from the SampleTreatmentReport
             object.
         *
         * @return geneId Identifier of the gene of the sample
         */
        public String getGeneId(){
                return this.sampleCM.getIdGene();
        }
        /**
         * Creates and sets the properties of a Query and its list of Exons.
         *
         * @param sequence Chain of characters that conform the complete Query
         * @param exons A list of strings containing the chain of characters of each
             Exon.
         */
        public void setQueryAndExons(String sequence, List<String> exons){
                /*Create SampleTreatmentReportCM data: query and exons*/
                Query q=new Query();
                List<Exon> eList=new ArrayList<Exon>();
```

```
                /*Fill STReportCM data*/
                q.setSequence(sequence); //query

                for(int i=0;i<exons.size();i++){//exons
                        Exon e=new Exon();
                        e.setNumber(BigInteger.valueOf(i));
                        e.setSequence(exons.get(i));
                        eList.add(e);
                }
                /*Add exons to query*/
                q.getExon().addAll(eList);
                /*Set query in the SampleTreatmentReportCM*/
                this.sampleCM.setQuery(q);
        }

        /**
         * Gets the properties of a Query.
         *
         * @return sequence Chain of characters that conform the complete Query
         */
        public String getQuery(){
                return this.sampleCM.getQuery().getSequence();
        }


        public List<HashMap<ExonProperties,Object>> getExons(){
                List<HashMap<ExonProperties,Object>> elist=new ArrayList<HashMap<
                    ExonProperties,Object>>();
                for(Exon e:this.sampleCM.getQuery().getExon()){
                        HashMap<ExonProperties,Object> exon=new HashMap<ExonProperties
                            ,Object>();
                        exon.put(ExonProperties.num,e.getNumber().intValue());
                        exon.put(ExonProperties.sequence,e.getSequence());
                        elist.add(exon);
                }
                return elist;
        }

        /**
         * Writes the SampleTreatmentReport into the corresponding XML file.
         * @param outputFileName
         */
        public void writeSampleTreatmentReportToFile(String outputFileName){
                this.writeContentToXMLFile(this.sampleCM, outputFileName);
        }
}
```

SequencherOutput.java, SequencherOutputTranslator.java

```
public class SequencherOutput extends SampleTreatmentOutput{
        String consensus;

        public SequencherOutput(String geneId) {
                super(geneId);
                consensus="";
        }
        /**
         * Sets the attribute consensus from a file that contains Sequencher data
         * @param inputFileName Name of the Sequencher data file
         */
        public void retrieveResultFromFile(String inputFileName) {
                try{
                        InputStream queryFile=new FileInputStream(inputFileName);
```

```
                        BufferedReader br = new BufferedReader (new InputStreamReader
                            (queryFile));

                        /*Read lines of the file and store data on the sequence
                            attribute*/
                        String aux;
                        while((aux=br.readLine())!=null){
                                this.consensus+=aux;
                        }
                        br.close();//Close file
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
public class SequencherOutputTranslator extends ResultTranslator{
        public SequencherOutputTranslator(SequencherOutput result){
                super(result);
        }
        /**
         * Converts data of source (expressed in Sequencher format) into a
         * SampleTreatmentResult
         *
         * Sequencher format:
         * - Character : to indicate that a nucleotide has not been sequenced
         * - Character N to indicate that a nucleotide has been deleted.
         * - Character A,C,T,G, and ambiguity codes to indicate the value of a
             nucleotide
         * @return report A SampleTreatmentReport containing the Sequencher data
         */
        public Report resultToReport() {
                SampleTreatmentReport report=new SampleTreatmentReport();
                /*1.IDGENE*/
                report.setGeneId(this.source.getGeneId());
                /*2.QUERY AND EXONS*/
                /*Separate Exons: Different pieces are separated by the character ':'.
                 * Example: 2 exons "ACA:::ACTG::" */
                String exons[]=((SequencherOutput)this.source).getConsensus().split("
                    :+");
                ArrayList<String> eList=new ArrayList<String>();
                for(int i=1;i<exons.length;i++){
                        /*a)Remove deletions: Deletion will be detected after
                            alignment*/
                        String seq=exons[i].replace("N", "");
                        /*b)Add string to the list*/
                        eList.add(seq);
                }
                report.setQueryAndExons(((SequencherOutput)this.source).getConsensus()
                    , eList);
                return report;
        }
}
```

org.pros.genoma.diagen.treatment.xmlReport

En este paquete se incluyen todas las clases java generadas automáticamente con JAXB: Exon.java, Query.java y SampleTreatmentReportCM.java

C.2. Fase Alineamiento

org.pros.genoma.diagen.alignment

Incluye las clases IAlignmentReportOperations.java, Alignment.java, AlignmentInput.java, AlignmentReport.java, AlignmentOutput.java, Exon.java, Query.java y Reference.java.

Alignment.java

```
/**
 * Class that executes the Alignment Phase:
 * 1. Receives the treated sample data from a previous phase in a
     SampleTreatmentReport
 * 2. An AlignmentInput is created
 * 3. The SampleTreatmentReport is translated to the AlignmentInput.
 * 4. Additional data (required to execute the alignment activities) is loaded from
     the GenomaDb.
 * 5. The Input data is exported into files with the suitable format.
 * 6. Activities to execute an alignment: Execute an alignment Algorithm that compares
     sequences.
 * 7. The output of these activities is saved in a file.
 * 8. An AlignmentResult is created
 * 9. The output data saved in a file is imported in the AlignmentResult
 * 10. An AlignmentReport is created (to exchange data with the next phase)
 * 11. The AlignmentResult is translated to a AlignmentReport
 *
 * Steps 1-5 are performed by the public method performInputTransformation()
 * Steps 6-9 are performed by the public method executeAlignment()
 * (8-9 by the private method retrieveAlignmentResult())
 * Steps 10-11 are performed by the public method performResultTransformation()
 *
 *
 * @author mariajo
 *
 */
public class Alignment {
public enum Algorithm {BLAST, SMWT, FLANK};

        private String geneId;
        private Algorithm algorithm;
        private GenomaDb db;
        private AlignmentInput input;
        private AlignmentOutput result;
        private String AlignmentOutputFileName;

        /**
         * @param geneId Gene Identifier of the analysis
         * @param Algorithm Algorithm/method used to performed the alignment
         * @param db A GenomaDb connection to obtain data required to perform the
             alignment
         */
        public Alignment(String geneId, Algorithm algorithm, GenomaDb db) {
                this.geneId=geneId;
                this.algorithm=algorithm;
                this.db=db;
        }


        /*------------------------PublicMethods-------------------*/
```

```
/**
 * Translates the SampleTreatmentReport into an AlignmentInput
 *
 * @param inputReport A SampleTreatmentReport containing treated sample data
 */
public void performInputTransformation(SampleTreatmentReport inputReport){
        InputTranslator translator=null;
        /*Choose application: Create the corresponding translator*/
        if(this.algorithm.compareTo(Algorithm.BLAST)==0){
                translator=new BlastInputTranslator(inputReport,db);
        }
        else if(this.algorithm.compareTo(Algorithm.SMWT)==0){
                translator=new SWInputTranslator(inputReport,db);
        }
        else if(this.algorithm.compareTo(Algorithm.FLANK)==0){
                translator=new FlankingInputTranslator(inputReport,db);
        }

        /*Translate report to input*/
        this.input=(AlignmentInput) translator.reportToInput();
}
/**
 * Executes the activities required to execute an alignment between
 * sequences and creates an AlignmentResult filled with the output
 * of those activities
 */
public void executeAlignment(){
        /*1. Input operations: Creates the files required to execute the
            alignment*/
        input.writeInputToFile();

        /*2. Executes alignment depending on the selected algorithm*/
        if(algorithm.compareTo(Algorithm.BLAST)==0){
                this.runBlast();
        }
        else if(algorithm.compareTo(Algorithm.SMWT)==0){
                this.runSmithWatterman();
        }
        else if(algorithm.compareTo(Algorithm.FLANK)==0){
                this.runFlankingAlgorithm();
        }
        /*3.Result operations: Creates and fills a Result with the output of
            the
         * alignment algorithm*/
        this.retrieveAlignmentResult();
}

/**
 * Translates the AlignmentResult into a AlignmentReport
 *
 * @return A AlignmentReport containing alignment data
 */
public AlignmentReport performResultTransformation(){
        AlignmentReport report=null;
        ResultTranslator translator=null;
        /*Choose algorithm: Create the corresponding translator*/
        if(this.algorithm.compareTo(Algorithm.BLAST)==0){
                translator=new BlastResultTranslator((BlastResult)result,(
                    BlastInput) this.input);
        }
        else if(this.algorithm.compareTo(Algorithm.SMWT)==0){
                translator=new SWResultTranslator((SWResult)result);
        }
        else if(algorithm.compareTo(Algorithm.FLANK)==0){
```

```
                translator=new FlankingResultTranslator((FlankingResult)
                    result);
            }
            /*Transformation: Execute the transformation*/
            report=(AlignmentReport)translator.resultToReport();
            return report;

    }
    /*------------------------------PrivateMethods-----------------*/

    /**
     * Executes the algorithm BLAST of NCBI.
     *
     */
    private void runBlast(){/*One blast for each exon*/
            this.AlignmentOutputFileName="BlastAlignmentOuput.xml";
            try {
                    String blastPath="C:/Program Files/NCBI/blast-2.2.23+/bin/";

                    String referenceFile=((BlastInput)this.input).
                        getFileNameReference();
                    String queryFile=((BlastInput)this.input).getFileNameQuery();
                    String referenceDatabase=((BlastInput)this.input).getReference
                        ().getIdSymbol(); //Name of database is the name of the
                        gene

                    /*Create database of the reference.
                     * Required to execute blastn.exe with the option of
                         ouputFormat=5(XML)*/
                    String commandDatabase=blastPath+"makeblastdb -in "+
                                    referenceFile+" -dbtype nucl -
                                        parse_seqids -out "+
                                        referenceDatabase;
                    Process p=Runtime.getRuntime().exec(commandDatabase);
                    p.waitFor();//Wait until creation of database

                    /*Execute BLAST: It performs an alignment for each exon of the
                         query.
                     *  The results are expressed in XML following the NCBI dtd.*/
                    String command=blastPath+"blastn.exe " +
                            "-db "+referenceDatabase + " -query "+queryFile+" -out
                                "+this.AlignmentOutputFileName+" -outfmt=5";
                    p=Runtime.getRuntime().exec(command);
                    p.waitFor();//Wait until the complete execution

            } catch (Exception e) {
                    e.printStackTrace();
            }
    }
    /**
     * Executes the algorithm Smith Watterman implemented by the GryCap
         collaboration.
     *
     */
    private void runSmithWatterman(){
            this.AlignmentOutputFileName="SWAlignmentOuput.xml";
            try {
                    String referenceFile=((SWInput)input).getFileNameReference();
                    String queryFile=((SWInput)input).getFileNameQuery();
                    /*Execute SmithWatterman*/
                    String args2[] =
                    {"./Alineador.jar","-r",referenceFile,"-g",this.geneId,"-c",
                        queryFile,"-o",this.AlignmentOutputFileName};
                    Alineador.main(args2);
            } catch (Exception e) {
```

```
                        e.printStackTrace();
                }
        }


        /**
         * Executes the algorithm based on flanking sequences implemented by the
             GryCap collaboration.
         *
         */
        private void runFlankingAlgorithm(){
                this.AlignmentOutputFileName="FlankingOutput.xml";
                /*File where the GenomaDb variations are loaded
                String variationsFileName=((FlankingAlgInput)input).
                    getFileNameVariations();
                /*Opeations to run algorithm*/
                /*....*///TODO call algorithm
                /*Operations to run algorithm*/
        }


        /**
         * Creates and fills an AlignmentResult with the output of the Alignment
         * execution.
         *
         * */
        private void retrieveAlignmentResult(){
                /*Choose algorithm: Create the corresponding Result*/
                if(algorithm.compareTo(Algorithm.BLAST)==0){
                        this.result=new BlastResult(this.geneId);
                        /*Result retrieval: load data from file*/
                        ((BlastResult)this.result).retrieveResultFromFile(this.
                            AlignmentOutputFileName);
                }
                else if(algorithm.compareTo(Algorithm.SMWT)==0){
                        this.result=new SWResult(this.geneId);
                        /*Result retrieval: load data from file*/
                        ((SWResult)this.result).retrieveResultFromFile(this.
                            AlignmentOutputFileName);
                }
        }
}
```

AlignmentReport.java

```
/**
 * AlignmentReport is a class that contains the implementation of
 * the conceptual model AlignmentReport.
 * It contains an attribute pointing the root object to the JAVA Objects
 * collection of the implementation.
 * The implementation of the conceptual model is performed with XMLSchema. In order
 * to work with XML and Java it has been used the JAXB technology .
 *
 * This class has X methods to create, set and get the properties of
 * these objects using this technology:
 * 1./2. Set/Get the identifier of the gene of the Alignment data
 * 3./4./5./6. Set/Get the reference/consensus sequence properties
 * 7./8 Set/Get the variations of the alignment
 *
 * @author mariajo
 *
 */
public class AlignmentReport extends Report implements IAlignmentReportOperations{
```

```
/**
 * @property alignmentCM a XML representation of the report using JAXB
 */
private AlignmentReportCM alignmentCM;


//The constructors are declared private to force the use of the create method
private AlignmentReport(String geneId){
        this.initializeReport();
        /*Set idGene in the AlignmentReportCM*/
        this.alignmentCM.setGeneId(geneId);
}

private AlignmentReport(AlignmentReportCM ar){
        this.initializeReport();
        this.alignmentCM = ar;
}

private void initializeReport(){
        this.alignmentCM = new AlignmentReportCM();
        this.alignmentCM.setVariations(new VariationList());
}

// TODO These static methods are intended to be object factory methods from
    the superclass Report

/**
 * Creates a new Alignment Report for a Gene
 * @param geneId the string symbol that identifies the gene (Example:"BRCA2")
 * @return the alignment report created
 * */
public static AlignmentReport create(String geneId) {
        return new AlignmentReport(geneId);
}

/**
 * Creates a new Alignment Report from a XML representation
 * @param ar an XML according to the Alignment Report XML Conceptual Model
 * @return the alignment report created
 * */
public static AlignmentReport createFromAR (AlignmentReportCM ar){
        return new AlignmentReport(ar);
}

/**
 * Creates an alignment Report with the variations stored in a database
 * This report can be used as input to look for these specific variation in
     the
 * alignment phase
 * @param geneId the string symbol that identifies the gene (Example:"BRCA2")
 * @return the alignment report created
 */
public static AlignmentReport createFromDB(String geneId){
        AlignmentReport ar =  new AlignmentReport(geneId);

        // TODO Connection must be created previously in the data access layer
            , this
        // must be changed
        GenomaDb db = new MySQLGenomaDb ();

        //Retrieve the gene and variation from the database
        Gene g = db.retrieveGeneById(geneId);
        List<PreciseVariation> gVariations = db.retrieveVariationsByGene(g);
```

```
            //Set the reference from the db info and the consensus as an external
                .txt file
            DnaFile dnaRef = ar.createDnaFile(g.getRefDB(),null,null, g.getId(),
                null, null);
            ar.setReference(dnaRef);

            //For each Precise Variation the XML equivalent is created and added
                to the Alignment report
            for(PreciseVariation pVar: gVariations){
                    String varType = pVar.getClass().getSimpleName();
                    DnaVariation xmlVar = new DnaVariation();

                    if (varType.equals("Insertion")){
                            ar.transformVariationToXML(pVar,xmlVar);
                            xmlVar.setValue(((Insertion) pVar).getInsBases());
                            ar.addInsVariation(xmlVar);
                    }

                    if (varType.equals("Indel")){
                            ar.transformVariationToXML(pVar,xmlVar);
                            xmlVar.setValue(((Indel) pVar).getInsBases());
                            ar.addSubVariation(xmlVar);
                    }

                    if (varType.equals("Deletion")){
                            DelVariation xmlDelVar = new DelVariation ();
                            ar.transformVariationToXML(pVar,xmlDelVar);
                            xmlDelVar.setValue(((Deletion) pVar).getDelBases());
                            BigInteger length = BigInteger.valueOf(((Deletion)
                                pVar).getDelLength());
                            xmlDelVar.setLength(length);
                            ar.addDelVaration(xmlDelVar);
                    }
                    /*Only for testing purpouses to check HGVS nomenclature
                    System.out.println(pVar.getHGVSgenomic());
                    */
            }
            return ar;
    }


    /**
     * Auxiliary method to set the common attributes of a JAXB Variation
     * @param pv the Precise Variation to be transformed
     * @param xmlVar the JAXB Variation object resulting from the transformation
     */
    private void transformVariationToXML (PreciseVariation pv, DnaVariation xmlVar
        ){
            xmlVar.setInitialPos(BigInteger.valueOf(pv.getStartpos()));
            xmlVar.setEndPos(BigInteger.valueOf(pv.getEndpos()));
            xmlVar.setHeterozygosis(false);
            xmlVar.setFsl(pv.getFsl());
            xmlVar.setFsr(pv.getFsr());
    }



    /*----------------Public Methods----------------*/
    /**
     * Creates a DnaFile JAXB object from a set of arguments
     * @param arguments the different attributes that compose a DNA Variation. If
         an argument
     * is not used it must be defined as null
     * @return A DnaFile containing the DNA Variation information
     */
```

```
public DnaFile createDnaFile(String id, BigInteger initialPos, BigInteger
    endPos,
                String refDb, String refFile,String sequence){
        DnaFile dnaFile=new DnaFile();

        if (id!=null) dnaFile.setId(id);
        if (initialPos != null) dnaFile.setInitialPos(initialPos);
        if (endPos != null) dnaFile.setEndPos(endPos);
        if (refDb != null) dnaFile.setRefDB(refDb);
        if (refFile != null) dnaFile.setRefFile(refFile);
        if (sequence != null) dnaFile.setSequence(sequence);

        return dnaFile;
}


/**
 * Creates a file with the properties of the AlignmentReport
 * @param outputFileName Name of the file
 */
public void writeAlignmentReportToFile(String outputFileName){
        this.writeContentToXMLFile(this.alignmentCM, outputFileName);
}

/*--------Result Methods: Set properties--------*/

/**
 * Sets the reference of the AlignmentReport JAXBObjects
 * @param reference A HashMap containing the attributes of the gene
 * */
public void setReference(DnaFile dnaRef){
        this.alignmentCM.setReference(dnaRef);
}
/**
 * Sets the consensus of the AlignmentReport JAXBObjects
 * @param consensus A HashMap containing the attributes of the gene
 * */
public void setConsensus(DnaFile dnaCons){
        this.alignmentCM.setConsensus(dnaCons);
}

public void addInsVariation(DnaVariation insVar){
        this.addVariationToList(insVar, "Ins");
}

public void addSubVariation(DnaVariation subVar){
        this.addVariationToList(subVar, "Sub");
}

public void addDelVaration(DelVariation delVar){
        this.addVariationToList(delVar, "Del");
}

private void addVariationToList(DnaVariation var, String type){
        List<JAXBElement<? extends DnaVariation>> vlist =  this.alignmentCM.
            getVariations().getInsOrDelOrSub();
        vlist.add(new JAXBElement<DnaVariation>(new QName("", type),
            DnaVariation.class,
                        VariationList.class, var));
        return;
}

/**
 * Sets the variations of the AlignmentReport JAXBObjects. This method is used
      by the BlastResult Translator
```

```
     * @param variations A List of HashMaps that containing the attributes of each
           variation
     * */
    public void setVariations(List<HashMap<VariationAttribute,Object>> variations)
        {
            VariationList variationList=new VariationList();
            /*Due JAXB restrictions it can not be created a list and
              set into the attribute that contains the list of variations.
               It can be solved retrieving the list(a) and adding the new objects(b
                   )*/
            List<JAXBElement<? extends DnaVariation>> vlist=variationList.
                getInsOrDelOrSub();//(a)
            for(HashMap<VariationAttribute,Object> v: variations){
                    /*Set the properties of the HashMap in a DnaVariation*/
                    DnaVariation variation=this.setVariation(v);
                    /*Add variation into the list*/
                    vlist.add(new JAXBElement<DnaVariation>(new QName("", (String)
                         v.get(VariationAttribute.type)), DnaVariation.class,
                                                 VariationList.class, variation
                                                     ));//(b)
            }
            this.alignmentCM.setVariations(variationList);
    }


    /*------------------InputMethods: Get properties-----------------------*/
    /**
     * Gets the gene of the AlignmentReport JAXBObjects.
     * @return The identifier of the gene
     * */
    public String getGeneId(){
            return this.alignmentCM.getGeneId();
    }
    /**
     * Gets the reference from the AlignmentReport JAXBObjects
     * @return A HashMap containing the properties of the gene reference
     */
    public HashMap<GeneAttribute,Object> getReference(){
            return this.getDnaFile(this.alignmentCM.getReference());
    }
    /**
     * Gets the consensus from the AlignmentReport JAXBObjects
     * @return A HashMap containing the properties of the gene consensus
     */
    public HashMap<GeneAttribute,Object> getConsensus(){
            return this.getDnaFile(this.alignmentCM.getConsensus());
    }

    /**
     * Gets the variations from the AlignmentReport JAXBObjects
     * @return A HashMap List containing the properties of each variation
     */
    public List<HashMap<VariationAttribute,Object>> getVariations(){
            List<HashMap<VariationAttribute,Object>> variationList=new ArrayList<
                HashMap<VariationAttribute,Object>>();
            List<JAXBElement<? extends DnaVariation>> vlist=this.alignmentCM.
                getVariations().getInsOrDelOrSub();

            for(JAXBElement<? extends DnaVariation> v: vlist){//For each variation
                 in the vlist
                    DnaVariation var=v.getValue();
                    /*Get the HashMap with the variation properties*/
                    HashMap<VariationAttribute,Object> variation=this.getVariation
                        (var);
                    /*Set additional properties of Variation*/
```

```
                variation.put(VariationAttribute.type, v.getName().
                    getLocalPart());//Type
                if(v.getName().getLocalPart().compareTo("Del")==0){//Length of
                     deletion
                        variation.put(VariationAttribute.length, ((
                            DelVariation)var).getLength().intValue());
                }
                /*Add HashMap to the list*/
                variationList.add(variation);
        }
        return variationList;
}


/*----------------------Private Methods-----------------------------------*/

/**
 * Sets a DnaVariation JAXB object from a HashMap
 * @param v A HashMap that contains the variation attributes
 * @return A DnaVariation containing the variation attributes
 */
private DnaVariation setVariation(HashMap<VariationAttribute,Object> v){
        DnaVariation variation=new DnaVariation();
        /*Deletion*/
        if("Del".compareTo((String) v.get(VariationAttribute.type))==0){
                variation=new DelVariation();
                if(v.get(VariationAttribute.length)!=null){
                        ((DelVariation) variation).setLength(BigInteger.
                            valueOf((Integer)(v.get(VariationAttribute.length))
                            ));
                }
        }

        /*Set general properties for all the types of variation*/
        if(v.get(VariationAttribute.initialPos)!=null){//initialPos
                variation.setInitialPos(BigInteger.valueOf((Integer)v.get(
                    VariationAttribute.initialPos)));
        }
        if(v.get(VariationAttribute.endPos)!=null){//endPos
                variation.setEndPos(BigInteger.valueOf((Integer)v.get(
                    VariationAttribute.endPos)));
        }
        if(v.get(VariationAttribute.value)!=null){//Value
                variation.setValue((String) v.get(VariationAttribute.value));
        }
        if(v.get(VariationAttribute.heterozygous)!=null){
                variation.setHeterozygosis((Boolean) v.get(VariationAttribute.
                    heterozygous));
        }
        if(v.get(VariationAttribute.fsl)!=null){
                variation.setFsl((String) v.get(VariationAttribute.fsl));
        }
        if(v.get(VariationAttribute.fsr)!=null){
                variation.setFsr((String) v.get(VariationAttribute.fsr));
        }

        return variation;
}

/**
 * Gets a HashMap from a DnaFile JAXB object
 * @param dnaFile A DnaFile that contains the Gene attributes
 * @return A HashMap containing the Gene attributes
 */
private HashMap<GeneAttribute,Object> getDnaFile(DnaFile dnaFile){
```

```
                HashMap<GeneAttribute,Object> dnaProperties=new HashMap<GeneAttribute,
                    Object>();

                /*Get general properties for all the types of variation*/
                dnaProperties.put(GeneAttribute.id, dnaFile.getId());
                if(dnaFile.getInitialPos()!=null){
                        dnaProperties.put(GeneAttribute.initialPos, dnaFile.
                            getInitialPos().intValue());
                }
                if(dnaFile.getEndPos()!=null){
                        dnaProperties.put(GeneAttribute.endPos, dnaFile.getEndPos().
                            intValue());
                }
                dnaProperties.put(GeneAttribute.refDb, dnaFile.getRefDB());
                dnaProperties.put(GeneAttribute.refFile, dnaFile.getRefFile());
                dnaProperties.put(GeneAttribute.sequence,dnaFile.getSequence());

                return dnaProperties;
        }

        /**
         * Gets a HashMap from a DnaVariation JAXB object
         * @param dnaVar A DnaFile that contains the Gene attributes
         * @return A HashMap containing the Gene attributes
         */
        private HashMap<VariationAttribute,Object> getVariation(DnaVariation dnaVar){
                HashMap<VariationAttribute,Object> variation=new HashMap<
                    VariationAttribute,Object>();

                /*Set general properties for all the types of variation*/
                variation.put(VariationAttribute.initialPos, dnaVar.getInitialPos().
                    intValue());
                variation.put(VariationAttribute.endPos, dnaVar.getEndPos().intValue()
                    );
                variation.put(VariationAttribute.value, dnaVar.getValue());
                variation.put(VariationAttribute.heterozygous, dnaVar.isHeterozygosis
                    ());
                variation.put(VariationAttribute.fsl, dnaVar.getFsl());
                variation.put(VariationAttribute.fsr, dnaVar.getFsr());

                return variation;
        }
}
```

Exon.java, Query.java, Reference.java

```
public class Exon {
        private int num;
        private String sequence;

        public Exon(int num, String sequence) {
                this.num = num;
                this.sequence = sequence;
        }
}

public class Query {
        public enum queryFormat {Sequencher}

        private String sequence;
        private List<Exon> exonList;

        public Query(){
                this.sequence="";
```

```
                this.exonList=new ArrayList<Exon>();
        }
}

public class Reference {
        public static enum WriteFormats{FASTA,TXT};
        private String idSymbol;
        private String accession;
        private String sequence;

        public Reference(String idSymbol) {
                this.idSymbol = idSymbol;

        }
        /*------------------------Public Methods--------------------------------*/
        /**
         * Sets reference attributes from the Genoma Database:
         *  the sequence and the accession number of NCBI.
         *
         * @param db Connection to the Genoma Database
         *
         */
        public void retrieveInfo(GenomaDb db){
                Reference r=db.retrieveReferenceById(this.idSymbol);
                this.accession=r.accession;
                this.sequence=r.sequence;
        }

        /**
         * Saves attributes of reference into a file.
         *
         * @param OutputfileName Path of the file to save data
         * @param format Format of the file to save data
         * */
        public void toFile(String outputFileName, WriteFormats format){
                if(format.compareTo(WriteFormats.FASTA)==0){

                }
                if(format.compareTo(WriteFormats.TXT)==0){
                        this.toTXTFile(outputFileName);
                }
        }

        /*------------------------PrivateMethods--------------------------------*/
        /**
         * Saves accession number and sequence of the reference following Fasta format
         *
         * @param outputFileName Path of the file to save data
         * */

        private void toTXTFile(String outputFileName){
                OutputStream referenceFile;
                try {
                        referenceFile = new FileOutputStream(outputFileName);
                        BufferedWriter bw = new BufferedWriter (new OutputStreamWriter
                             (referenceFile));
                        /*Write sequence in TXT format*/
                        bw.write(this.sequence);
                        bw.close();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

org.pros.genoma.diagen.alignment.xmlReport

En este paquete se incluyen todas las clases java generadas automáticamente con JAXB: DnaFile.java, DnaVariation.java, DelVariation.java, VariationList.java y AlignmentReportCM.java

org.pros.genoma.diagen.alignment.blast

Incluye las clases BlastInput.java, BlastInputTranslator.java, BlastOutput.java y BlastOutputTranslator.java.

BlastInput.java, BlasInputTranslator.java

```
public class BlastInput extends AlignmentInput{
        Reference reference;

        String fileNameReference;
        String fileNameQuery;

        public BlastInput(String geneId) {
                super(geneId);
                this.reference=new Reference(geneId);
                this.fileNameReference="BlastReference.fasta";
                this.fileNameQuery="BlastQuery.txt";
        }

        public Reference getReference() {
                return reference;
        }
        public String getFileNameReference() {
                return fileNameReference;
        }
        public void setFileNameReference(String fileNameReference) {
                this.fileNameReference = fileNameReference;
        }
        public String getFileNameQuery() {
                return fileNameQuery;
        }
        public void setFileNameQuery(String fileNameQuery) {
                this.fileNameQuery = fileNameQuery;
        }

        /*------------------Public Methods-------------------------------*/
        /**
         * Retrieves reference information from the Genoma Db
         * @param db A GenomaDb connection to the database
         */
        public void retrieveReferenceFromGenoma(GenomaDb db){
                this.reference.retrieveInfo(db);
        }
        /**
         * Loads Reference to FASTA file and Query to TXT file. Required to execute
             blastn.exe
         * */
        public void writeInputToFile(){
                try {
                        /*Reference to File*/
```

```
                        OutputStream referenceFile = new FileOutputStream(this.
                            fileNameReference);
                        BufferedWriter bwr = new BufferedWriter (new
                            OutputStreamWriter (referenceFile));
                        /*Write sequence in FASTA format*/
                        bwr.write(">"+this.reference.getAccession()+"\n");
                        bwr.write(this.reference.getSequence());
                        bwr.close();

                        /*Query To File*/
                        OutputStream queryFile= new FileOutputStream(this.
                            fileNameQuery);
                        BufferedWriter bwq = new BufferedWriter (new
                            OutputStreamWriter (queryFile));
                        /*Write list of exons into a file: It stores each exon in a
                            line preceded by
                         *'>'+iteration_num. This concrete format is required to
                             execute the alignment
                         * algorithm blast for each exon separately.*/
                        for(int i=0;i<this.query.getExonList().size();i++){
                                bwq.write(">"+i+"\r\n"+this.query.getExon(i).
                                    getSequence()+"\r\n");
                        }
                        bwq.close();

                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}

public class BlastInputTranslator extends InputTranslator {
        GenomaDb db;

        public BlastInputTranslator(SampleTreatmentReport source, GenomaDb db) {
                super(source);
                this.db=db;
        }

        /*A sampleTreatment Report is translated to a BlastInput*/
        public Input reportToInput() {
                BlastInput input=new BlastInput(((SampleTreatmentReport)this.source).
                    getGeneId());
                /*Retrieve Reference*/
                input.retrieveReferenceFromGenoma(db);
                /*Retrieve query and exons*/
                //Sequence of query
                input.getQuery().setSequence(((SampleTreatmentReport)this.source).
                    getQuery());
                //List sequence of exons
                List<Exon> exonList=new ArrayList<Exon>();
                for(HashMap<ExonProperties,Object> e: ((SampleTreatmentReport)this.
                    source).getExons()){
                        Exon exon=new Exon((Integer)e.get(ExonProperties.num),(String)
                            e.get(ExonProperties.sequence));
                        exonList.add(exon);
                }
                input.getQuery().setExonList(exonList);

                return input;
        }
}
```

BlastOutput.java, BlastOuputTranslator.java

```
public class BlastOutput extends AlignmentOutput {
        private BlastOutput blastOuput;
        private ArrayList<Difference> differencesList;

        public BlastOutput(String idGene) {
                super(idGene);
                this.differencesList=new ArrayList<Difference>();
        }

        /**
         * Retrieves blastAlignment from file that follows BlastNCBI dtd format
         * and sets the differenceList.
         *
         * @param XMLInputFileName Path of the XMLfile that contains Blast alignment
             results
         * */
        public void retrieveResultFromFile(String inputFileName){
                this.blastOuput=(BlastOutput)this.getContentFromXMLFile(BlastOutput.
                    class, inputFileName, null);
                /*Set BlastResult Attributes*/
                this.calculateDifferenceList();
        }

        /*-------------------------PrivateMethods---------------------*/
        /**
         * Calculates the differencesList from blastAlignment object:
         *    1. Covers all iterations: For each iteration gets the first hit (best
             match)
         *    and calculates the differences among the aligned reference and query.
         *
         *    2.After getting differences it joins the consecutive ones.
         * */
        private void calculateDifferenceList(){
                /*1. Retrieve differences from each iteration*/
                List<Iteration> iterations=this.blastOuput.getBlastOutputIterations().
                    getIteration();
                for(Iteration it: iterations){
                        /*Get differences from first Hsp*/
                        Hsp hsp=it.getIterationHits().getHit().get(0).getHitHsps().
                            getHsp().get(0);
                        this.differencesList.addAll(getDifferencesFromHsps(hsp));
                }
                /*2. Join in one difference the differences that are consecutive*/
                this.joinConsecutiveDifferences();
        }
        /**
         * Calculates the differences among the aligned reference and query
         * from a hsp of a hit.
         *
         * @param hsp Hsp object of blast that contains an alignments for the
             reference and the query
         * @return differenceList A list containing the differences found
         *
         * @see Hsp from package NCBIBlastFORMAT
         * */
        private List<Difference> getDifferencesFromHsps(Hsp hsp){
                List<Difference> differenceList=new ArrayList<Difference>();

                /*Blast Alignment data: query, subsegment of reference and subsegment
                    start regarding reference*/
                String query=hsp.getHspQseq();
                String reference=hsp.getHspHseq();
                int reference_from=Integer.valueOf(hsp.getHspHitFrom());
```

```
            int reference_shift=0; //Counter of positions that are shifted in the
                reference while blast alignment

            if(hsp.getHspMidline().contains("␣")){//a space in midline means a
                mismatch
                    for(int i=0; i<reference.length();i++){ //Add 1 to all
                        positions: Here index starts in 0
                            char cref=reference.charAt(i);
                            char cquery=query.charAt(i);
                            int pos=reference_from+i-reference_shift;
                            if(cref!=cquery){
                                    if(cref=='-') {           //INSERTION: It is
                                        saved the value inserted in the query
                                            differenceList.add(new Difference(pos
                                                -1,pos,String.valueOf(cquery),
                                                Difference.Type.Ins));
                                            reference_shift++; //Each time an
                                                insertion appears the reference is
                                                shifted 1 position
                                    }
                                    else if(cquery=='-'){ //DELETION: it is saved
                                        the value deleted from the reference
                                            differenceList.add(new Difference(pos,
                                                pos,String.valueOf(cref),Difference
                                                .Type.Del));
                                    }
                                    else {                                  //
                                        SUBSTITUTION
                                            differenceList.add(new Difference(pos,
                                                pos,String.valueOf(cquery),
                                                Difference.Type.Sub));
                                    }
                            }
                    }//For
            }//If
            return differenceList;
    }
    /**
     * Join the consecutive differences of Length 1 of the same type into a unique
          difference.
     * The first difference is updated and the consecutive ones
     * are deleted from the differenceList.
     *
     * 1. Join insertions: both start in the same position
     * 2. Join deletions and substitutions: the second one starts in the next
         position the previous one ends
     * */
    private void joinConsecutiveDifferences(){
            for(int i=1; i<differencesList.size();i++){
                    /*Consecutive differences of same type*/
                    if(differencesList.get(i).getType()==differencesList.get(i-1).
                        getType()){
            /*1*/ if((differencesList.get(i).getType().compareTo(Difference.Type.
                Ins)==0 &&
                            differencesList.get(i).getInitialPosition()==
                                differencesList.get(i-1).getInitialPosition())
                                                    ||
            /*2*/     (differencesList.get(i).getType().compareTo(Difference.Type.
                Ins)!=0 /*Deletion and substitution*/ &&
                             differencesList.get(i).getInitialPosition()==
                                 differencesList.get(i-1).getEndPosition()+1)
                            ){
                                    //Join (concat new base and change end
                                        position) and remove
```

```
                                        differencesList.get(i-1).setBases(
                                            differencesList.get(i-1).getBases().concat(
                                            differencesList.get(i).getBases()));
                                        differencesList.get(i-1).setEndPosition(
                                            differencesList.get(i).getEndPosition());
                                        differencesList.remove(i);
                                        i--;//Due the deletion
                                    }
                                }//If (equal type)
                            }//For
                    }
            }

            public class BlastOutputTranslator extends ResultTranslator {
                    BlastInput input;


                    public BlastOutputTranslator(BlastOutput source, BlastInput input) {
                            super(source);
                            this.input=input;
                    }

                    @SuppressWarnings("unchecked")
                    public Report resultToReport() {
                            AlignmentReport report=AlignmentReport.create(this.source.getGeneId())
                                ;

                            /*Fill data from Blast output and BlastInput data*/
                            //Reference
                            String id = ((BlastOutput)this.source).getBlastOuput().
                                getBlastOutputQueryID();
                            String refDb = input.getReference().getAccession();
                            String refFile = input.getFileNameReference();
                            String sequence = input.getReference().getSequence();

                            DnaFile refDNA = report.createDnaFile(id, null, null, refDb, refFile,
                                sequence);
                            report.setReference(refDNA);

                            //Consensus
                            refFile = input.getFileNameQuery();
                            sequence = input.getQuery().getSequence();

                            DnaFile consDNA = report.createDnaFile(null, null, null, null, refFile
                                , sequence);
                            report.setConsensus(consDNA);

                            /*Fill data from BlastResult*/
                            //Variations
                            List<HashMap<VariationAttribute,Object>> variations=new ArrayList<
                                HashMap<VariationAttribute,Object>>();
                            for(Difference difference: ((BlastOutput)this.source).
                                getDifferencesList()){
                                    HashMap var=new HashMap();
                                    var.put(VariationAttribute.initialPos, difference.
                                        getInitialPosition());
                                    var.put(VariationAttribute.endPos, difference.getEndPosition()
                                        );
                                    var.put(VariationAttribute.value, difference.getBases());
                                    if(difference.getType().compareTo(Difference.Type.Ins)==0){
                                            var.put(VariationAttribute.type, "Ins");
                                    }
                                    else if(difference.getType().compareTo(Difference.Type.Del)
                                        ==0){
                                            var.put(VariationAttribute.type, "Del");
```

```
                        var.put(VariationAttribute.length, difference.
                            getLength());
                    }
                    else if(difference.getType().compareTo(Difference.Type.Sub)
                        ==0){
                        var.put(VariationAttribute.type, "Ind");
                    }
                var.put(VariationAttribute.initialPos, difference.getInitialPosition()
                    );
                this.setVariationFlankingSequences(difference, var);
                this.setVariationHeterocigosis(difference, var);
                variations.add(var);
                }
                report.setVariations(variations);

                return report;
        }

        private void setVariationFlankingSequences(Difference difference,HashMap<
            VariationAttribute,Object> var){
                String sequence=this.input.getReference().getSequence();
                /*Set flanking sequences*/
                if(difference.getType().compareTo(Difference.Type.Ins)==0){
                        var.put(VariationAttribute.fsl, sequence.substring(difference.
                            getInitialPosition()-20, difference.getInitialPosition()));
                        var.put(VariationAttribute.fsr, sequence.substring(difference.
                            getEndPosition()-1,difference.getEndPosition()+19));
                }//Falta cambiar -1
                else if(difference.getType().compareTo(Difference.Type.Sub)==0 ||
                    difference.getType().compareTo(Difference.Type.Del)==0){
                        var.put(VariationAttribute.fsl, sequence.substring(difference.
                            getInitialPosition()-21, difference.getInitialPosition()-1)
                            );
                        var.put(VariationAttribute.fsr, sequence.substring(difference.
                            getEndPosition(),difference.getEndPosition()+20));
                }
        }

private void setVariationHeterocigosis(Difference difference,HashMap<
    VariationAttribute,Object> var){
        String sequence=this.input.getReference().getSequence();
        if(difference.getType().compareTo(Difference.Type.Sub)==0){
                /*Set heterocygous attribute and the value that has really changed*/
                if(difference.getBases().matches("[ATGCatgc]+")==false){
                        /*Obtain both bases represented by ambiguous base*/
                        String bases=this.getBases(difference.getBases());
                        if(bases!=null){
                                /*Check which nucleotide is different than the
                                    reference*/
                                if(bases.charAt(0)!=sequence.charAt(difference.
                                    getInitialPosition()-1)){
                                        var.put(VariationAttribute.value, String.
                                            valueOf(bases.charAt(0)));
                                }else {
                                        var.put(VariationAttribute.value, String.
                                            valueOf(bases.charAt(1)));
                                }
                                var.put(VariationAttribute.heterozygous, true);//
                                    Establish heterozygous to true;
                        }else {//TEMPORARY
                                var.put(VariationAttribute.value,difference.getBases()
                                    );
                                var.put(VariationAttribute.heterozygous,false);
                        }
                }//If Heterozygous
```

```
        }//If Substitution
}

/**
 * Table containing ambiguous bases codes
 */
        @SuppressWarnings("serial")
        protected String getBases(String UIBCode){
                final HashMap<String, String> UIBTable=new HashMap<String, String>(){
                        {
                                put("K", "GT");
                                put("M", "AC");
                                put("R", "AG");
                                put("S", "CG");
                                put("W", "AT");
                                put("Y", "CT");
                        }
                };
                return UIBTable.get(UIBCode.toUpperCase());
        }
}
```

org.pros.genoma.diagen.alignment.sw

Incluye las clases SWInput.java, SWTranslator.java, SWOutput.java y SWOutputTranslator.java.

SWInput.java, SWTranslator.java

```
public class SWInput extends AlignmentInput {
        Reference reference;

        String fileNameReference;
        String fileNameQuery;

        public SWInput(String geneId) {
                super(geneId);
                this.reference=new Reference(geneId);
                this.fileNameReference="SWReference.fasta";
                this.fileNameQuery="SWQuery.txt";
        }

        /*-------------------------------Public Methods-------------------------------*/
        /**
         * Retrieves reference information from the Genoma Db
         * @param db A GenomaDb connection to the database
         */
        public void retrieveReferenceFromGenoma(GenomaDb db){
                this.reference.retrieveInfo(db);
        }
        /*Load Reference and Query to TXT file with a specific format*/
        public void writeInputToFile() {
                try {
                        /*Reference to File*/
                        OutputStream referenceFile = new FileOutputStream(this.
                            fileNameReference);
                        BufferedWriter bwr = new BufferedWriter (new
                            OutputStreamWriter (referenceFile));
                        /*Write sequence*/
```

```
                        bwr.write(">"+this.reference.getAccession()+"\n"+this.
                            reference.getSequence());
                        bwr.close();

                        /*Query To File*/
                        OutputStream queryFile= new FileOutputStream(this.
                            fileNameQuery);
                        BufferedWriter bwq = new BufferedWriter (new
                            OutputStreamWriter (queryFile));
                        /*Write list of exons separated by ':'*/
                        for(Exon e: this.query.getExonList()){
                                bwq.write(":"+e.getSequence()+":");
                        }
                        bwq.close();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}

public class SWInputTranslator extends InputTranslator{
        GenomaDb db;

        public SWInputTranslator(Report source, GenomaDb db) {
                super(source);
                this.db=db;
        }

        public Input reportToInput() {
                SWInput input=new SWInput(((SampleTreatmentReport)this.source).
                    getGeneId());
                /*Retrieve Reference*/
                input.retrieveReferenceFromGenoma(db);
                /*Retrieve query and exons*/
                //Sequence of query
                input.getQuery().setSequence(((SampleTreatmentReport)this.source).
                    getQuery());
                //List sequence of exons
                List<Exon> exonList=new ArrayList<Exon>();
                for(HashMap<ExonProperties,Object> e: ((SampleTreatmentReport)this.
                    source).getExons()){
                        Exon exon=new Exon((Integer)e.get(ExonProperties.num),(String)
                            e.get(ExonProperties.sequence));
                        exonList.add(exon);
                }
                input.getQuery().setExonList(exonList);

                return input;
        }
}
```

SWOutput.java, SWOutputTranslator.java

```
public class SWOutput extends AlignmentOutput {
        AlignmentReportCM alignmentCM;

        public SWOutput(String geneId){
                super(geneId);
                this.alignmentCM=new AlignmentReportCM();
        }

        public AlignmentReportCM getAlignmentCM() {
                return alignmentCM;
        }
```

```
        /**
         * Loads result data from a file. Assumptions: data is gathered in one file
         * @param inputFileName Path of the file that contains the results of
             alignment
         */
        public void retrieveResultFromFile(String inputFileName) {
                this.alignmentCM=(AlignmentReportCM) this.getContentFromXMLFile(
                    AlignmentReportCM.class, inputFileName, "AlignmentReport.xsd");
        }

}

public class SWOutputTranslator extends ResultTranslator {

        public SWOutputTranslator(SWOutput source) {
                super(source);
        }

        public Report resultToReport() {
                return AlignmentReport.createFromAR(((SWOutput)this.source).
                    getAlignmentCM());
        }
}
```

org.pros.genoma.diagen.alignment.flanking

Incluye las clases FlankingInput.java, FlankingTranslator.java, FlankingOutput.java y FlankingOutputTranslator.java.

FlankingInput.java, FlankingTranslator.java

```
public class FlankingInput extends AlignmentInput{
        AlignmentReportCM alignmentReportCM;
        String fileNameVariations;


        public FlankingInput(String geneId) {
                super(geneId);
                this.alignmentReportCM=new AlignmentReportCM();
                this.fileNameVariations="FlankingAlignmentReport.xml";
        }


        public String getFileNameVariations() {
                return this.fileNameVariations;
        }

        /**
         * Retrieves reference information from the Genoma Db
         * @param db A GenomaDb connection to the database
         */

        public void retrieveAligmentReportFromGenomaDb(GenomaDb db){
                //1. llamada a genomaDb: db.getAllVariationsFromId(geneId) o algo así
                //2. El método devolvera una lista de PreciseVariations?
                //3. Instanciar Objetos JAXB
                //GeneId
                this.alignmentReportCM.setGeneId(this.getGeneId());
                /*Implementar llamadas para cargar las variaciones de genomadb
                 * e introducirlas en el alignmentReport
```

```
                *

                * */
        }
        /**
         * Writes the AlignmentReport to a XML file
         * in order to create the input of the flanking algorithm
         */
        public void writeInputToFile() {

        }

}
public class FlankingInputTranslator extends InputTranslator{
        GenomaDb db;
        public FlankingInputTranslator(Report source,GenomaDb db) {
                super(source);
                this.db=db;
        }


        public Input reportToInput() {
                FlankingInput input=new FlankingInput(((SampleTreatmentReport)this.
                    source).getGeneId());
                /*Retrieve Variations from Genoma*/
                input.retrieveAligmentReportFromGenomaDb(db);

                /*Retrieve query and exons*/
                //Sequence of query
                input.getQuery().setSequence(((SampleTreatmentReport)this.source).
                    getQuery());
                //List sequence of exons
                List<Exon> exonList=new ArrayList<Exon>();
                for(HashMap<ExonProperties,Object> e: ((SampleTreatmentReport)this.
                    source).getExons()){
                        Exon exon=new Exon((Integer)e.get(ExonProperties.num),(String)
                            e.get(ExonProperties.sequence));
                        exonList.add(exon);
                }
                input.getQuery().setExonList(exonList);

                return input;
        }
}
```

FlankingOutput.java y FlankingOutputTranslator.java.

```
public class FlankingOutput extends AlignmentOutput{
        AlignmentReportCM alignmentReportCM;
        public FlankingOutput(String idGene) {
                super(idGene);
                this.alignmentReportCM=new AlignmentReportCM();
        }

        public AlignmentReportCM getAlignmentReportCM(){
                return this.alignmentReportCM;
        }


        public void retrieveResultFromFile(String inputFileName) {
                this.getContentFromXMLFile(this.alignmentReportCM.getClass(),
                    inputFileName, null);
        }
        public void writeResultToFile(String outputFileName) {
```

```
                this.writeContentToXMLFile(this.alignmentReportCM,outputFileName);
        }

}

public class FlankingOutputTranslator extends ResultTranslator {

        public FlankingOutputTranslator(FlankingOutput source) {
                super(source);
                // TODO Auto-generated constructor stub
        }

        public Report resultToReport() {
                return AlignmentReport.createFromAR(((FlankingOutput) this.source).
                    getAlignmentReportCM());
        }
}
```

C.3. Fase Conocimiento

org.pros.genoma.diagen.knowledge

Incluye las clases Knowledge.java, KnowledgeInput.java, KnowledgeOutput, HGDBInput.java, HGDBInputTranslator.java, HGDBOutput.java, HGDBOuputTranslator.java, Gene.java, Exon.java, PreciseVariation.java, Insertion.java, Deletion.java, Indel.java, Documentation.java, Bibliography.java, Transcript.java y Aminoacid.java.

Knowledge.java

```
public class Knowledge {
        public static enum Strategy{POS,FLANKING,BOTH};
        private String geneId;
        private Input input;
        private Result result;
        private Strategy strategy;

        private List<PreciseVariation> variations;

        GenomaDb db;

        public Knowledge(String geneId,Strategy strategy, GenomaDb db){
                this.geneId=geneId;
                this.strategy=strategy;
                this.db=db;
        }

        /*-----------------------PublicMethods--------------------*/
        /**
         * Translates the SampleTreatmentReport into an AlignmentInput
         *
         * @param inputReport A SampleTreatmentReport containing treated sample data
         */
        public void performInputTransformation(AlignmentReport inputReport){
                InputTranslator translator=null;
                        translator=new HGDBInputTranslator(inputReport);
                /*Translate report to input*/
```

```
            this.input=(HGDBInput) translator.reportToInput();
        }

        public void executeVariationKnowdlege(){
            /*1. Input operations: Sets the objects required to execute the
                variationKnowdlege*/
            this.variations=((HGDBInput)this.input).getVariations();
            /*2. Execute the location depending on search strategy*/
            if(strategy.compareTo(Strategy.POS)==0){
                    this.locateVariationsByPos();
            }
            else if(strategy.compareTo(Strategy.FLANKING)==0){
                    this.locateVariationsByFlanking();
            }
            else if(strategy.compareTo(Strategy.BOTH)==0){
                    this.locateVariationsByBoth();
            }
            /*Result operations: Creates and fills a Result with the output of the
             * alignment algorithm*/
            this.retrieveVariationKnowdlegeResult();
        }

        /**
         * Translates the VariationKnowdlegeResult into a VariationKnowdlegeReport
         *
         * @return A VariationKnowdlegeReport containing alignment data
         */
        public VariationKnowdlegeReport performResultTransformation(){
            VariationKnowdlegeReport report=null;
            ResultTranslator translator=null;

            translator=new HGDBOutputTranslator((HGDBOutput) result, (HGDBInput)
                this.input);

            /*Transformation: Execute the transformation*/
            report=(VariationKnowdlegeReport)translator.resultToReport();
            return report;

        }

        /*--------------------Private Methods-----------------------------------*/
        private void locateVariationsByPos(){
            for (PreciseVariation pv:this.variations){
                    pv.documentVariationByPos(this.db);
            }
        }
        private void locateVariationsByFlanking(){
            for (PreciseVariation pv:this.variations){
                    pv.documentVariationByFlanking(this.db);
            }
        }
        private void locateVariationsByBoth(){
            for (PreciseVariation pv:this.variations){
                    pv.documentVariationByPos(this.db);
                    pv.documentVariationByFlanking(this.db);
            }
        }
        private void retrieveVariationKnowdlegeResult(){
            this.result=new HGDBOutput(geneId);
            /*Result retrieval: load data from file*/
            ((HGDBOutput)this.result).retrieveResultFromObject(variations);
        }
}
```

HGDBInput.java, HGDBInputTranslator.java

```
public class HGDBInput extends KnowdlegeInput{
        private Gene reference;
        private Gene consensus;
        private List<PreciseVariation> variations;

        public HGDBInput(String geneId) {
                super(geneId);
        }
}

public class HGDBInputTranslator extends InputTranslator {

        public HGDBInputTranslator(AlignmentReport source) {
                super(source);
        }

        public Input reportToInput() {
                String geneId=((AlignmentReport)this.source).getGeneId();
                HGDBInput input=new HGDBInput(geneId);

                /*Reference*/
                HashMap<GeneAttribute,Object> ref=((AlignmentReport)this.source).
                    getReference();
                String id= (String) ref.get(GeneAttribute.id);
                String sequence=(String) ref.get(GeneAttribute.sequence);
                String refDb=(String) ref.get(GeneAttribute.refDb);
                String filePath=(String) ref.get(GeneAttribute.refFile);
                Gene reference=new Gene(geneId, id, sequence, refDb, filePath);

                input.setReference(reference);

                /*Consensus*/
                HashMap<GeneAttribute,Object> cons=((AlignmentReport)this.source).
                    getConsensus();
                id= (String) cons.get(GeneAttribute.id);
                sequence=(String) cons.get(GeneAttribute.sequence);
                refDb=(String) cons.get(GeneAttribute.refDb);
                filePath=(String) cons.get(GeneAttribute.refFile);
                Gene consensus=new Gene(geneId, id, sequence, refDb, filePath);

                input.setConsensus(consensus);

                /*VariationList*/
                List<PreciseVariation> pvList=new ArrayList<PreciseVariation>();
                List<HashMap<VariationAttribute,Object>> variations=((AlignmentReport)
                    this.source).getVariations();
                for(HashMap<VariationAttribute,Object> variation: variations){
                        int startpos=(Integer)variation.get(VariationAttribute.
                            initialPos);
                        int endpos=(Integer) variation.get(VariationAttribute.endPos);
                        boolean heterozygous=(Boolean)variation.get(VariationAttribute
                            .heterozygous);
                        String fsl=(String) variation.get(VariationAttribute.fsl);
                        String fsr=(String) variation.get(VariationAttribute.fsr);
                        String value=(String) variation.get(VariationAttribute.value);

                        PreciseVariation pv=null;
                        if(((String)variation.get(VariationAttribute.type)).compareTo(
                            "Ins")==0){
                                pv=new Insertion(startpos,endpos,value,reference,fsr,
                                    fsl);
                        }
```

```
                        else if(((String)variation.get(VariationAttribute.type)).
                            compareTo("Del")==0){
                                int length= (Integer) variation.get(VariationAttribute
                                    .length);
                                pv=new Deletion(startpos,endpos,length,reference,fsr,
                                    fsl);
                        }
                        else if(((String)variation.get(VariationAttribute.type)).
                            compareTo("Ind")==0){
                                pv=new Indel(startpos,endpos,value,reference,fsr,fsl);
                        }
                        //Heterozygous cannot be introduced using the constructors
                            because is an optional property
                        pv.setHeterozygous(heterozygous);
                        pvList.add(pv);
                }
                input.setVariations(pvList);

                return input;
        }
}
```

HGDBOutput.java, HGDBOuputTranslator.java

```
public class HGDBOutput extends KnowdlegeOutput {
        List<PreciseVariation> variations;

        public HGDBOutput(String geneId) {
                super(geneId);
        }
        public void retrieveResultFromObject(Object inputObject) {
                        this.variations=(List<PreciseVariation>) inputObject;
        }
}

public class HGDBOutputTranslator extends ResultTranslator{
        HGDBInput input;
        public HGDBOutputTranslator(HGDBOutput source, HGDBInput input) {
                super(source);
                this.input=input;
        }
        public Report resultToReport() {
                VariationKnowdlegeReport report=new VariationKnowdlegeReport();

                report.setGeneId(((HGDBOutput)this.source).getGeneId());

                //Reference
                HashMap<GeneAttribute,Object> reference=new HashMap<GeneAttribute,
                    Object>();
                reference.put(GeneAttribute.id, input.getReference().getId());
                reference.put(GeneAttribute.refDb, input.getReference().getRefDB());
                reference.put(GeneAttribute.refFile,input.getReference().getFilePath()
                    );
                reference.put(GeneAttribute.sequence,input.getReference().getSequence
                    ());
                report.setReference(reference);
                //Consensus
                HashMap<GeneAttribute,Object> consensus=new HashMap<GeneAttribute,
                    Object>();
                consensus.put(GeneAttribute.id,input.getConsensus().getId());
                consensus.put(GeneAttribute.refFile,input.getConsensus().getFilePath()
                    );
                consensus.put(GeneAttribute.sequence,input.getConsensus().getSequence
                    ());
```

```
                report.setConsensus(consensus);

                //Variations
                //TODO code to set variations and documentation with simple types
                List<PreciseVariation> variations=((HGDBOutput)this.source).
                    getVariations();
                report.setDependentVariations(variations);

                return report;
        }
}
```

PreciseVariation.java, Insertion.java, Deletion.java, Indel.java

```
/**
 * The PreciseVariations class Represents the attributes that have all kind of
     variations
 * and implements the required methods to locate them in a Genoma Database:
 *
 * 1. Searches by position: The variation and its synonym variations are searched in
     the database
 * by checking its gene, its POSITION, its type and the nucleotides changed.
 * 2. Searches by flanking: The variation is searched in the database by checking its
     gene,
 * its FLANKING sequences, its type and the nucleotides changed.
 *
 * @author mariajo
 *
 */

public abstract class PreciseVariation {
        protected int startpos;
        protected int endpos;
        protected boolean heterozygous;
        protected int heterozygousFreq;//Field not filled
        protected String fsr;
        protected String fsl;


        protected Gene refGene;
        private List<Documentation> documentationList;

        public PreciseVariation(int startpos, int endpos, Gene refGene, String fsr,
            String fsl) {
                this.startpos = startpos;
                this.endpos = endpos;
                this.refGene=refGene;
                this.documentationList=new ArrayList<Documentation>();
                this.fsr=fsr;
                this.fsl=fsl;
        }
        /*-------Methods to work with lists----------------------*/
        public void addDocumentation(Documentation documentation) {
                this.documentationList.add(documentation);
        }
        public void appendDocumentation(List<Documentation> list){
                if(list!=null){
                        if(this.documentationList.isEmpty()){//New list
                                this.documentationList=list;
                        }
                        else{//Append new elements
                                this.documentationList.addAll(list);
                        }
                }
```

```
        }
        /*----------------------Abstract methods-----------------------------------------*/
        /*Different method for each kind of variation*/
        /**
         * Searches the a variation By its position in a GenomaDb and retrieves
         * its properties, the associated phenotype and the associated documentation
         * @param db Connection to GenomaDb
         */
        public abstract void documentVariationByPos(GenomaDb db);

        public abstract String getHGVSgenomic();

        public abstract String getHGVScoding();

        //public abstract String getHGVSprotein();*/


        /*----------------------Public Methods-------------------------------------------*/

        /*Common method for all the types of PreciseVariation*/
        /**
         * Searches a variation in a GenomaDb by comparing their flanking sequences
         * @param db Connection to GenomaDb
         */
        public void documentVariationByFlanking(GenomaDb db){
                if(this.fsl!=null && this.fsr!=null){
                        this.appendDocumentation(db.checkVariationByFlanking(this,new
                            Float(0.9)));
                }else{
                        String error="Error searching by flanking: Flanking sequences
                            not included";
                        System.err.println(error);
                }
        }
}


public class Insertion extends PreciseVariation {
        private String insBases;

        public Insertion(int startpos, int endpos, String insBases, Gene refGene,
            String fsr, String fsl) {
                super(startpos, endpos, refGene,fsr,fsl);
                this.insBases=insBases;
        }

        /*----------------------Public Methods-------------------------------------------*/
        public void documentVariationByPos(GenomaDb db) {
                /*Search the variation in HGDB*/
                this.appendDocumentation(db.checkVariationByPos(this));

                /*Get synonymous variations and search them on HGDB*/
                List<Insertion> synonymousInsertions=this.getSynonymousInsertions();
                for(Insertion i:synonymousInsertions){
                        List<Documentation> auxList;
                        auxList=db.checkVariationByPos(i);
                        if(auxList!=null && !auxList.isEmpty()){
                                this.appendDocumentation(auxList);
                        }
                }
        }

        /*----------------------Private Methods------------------------------------------*/
        private List<Insertion> getSynonymousInsertions() {/*Code to calculate
            synonymous insertions*/
```

```
                List<Insertion> synonymous = new ArrayList<Insertion>();

                //TODO: solve out of array index exceptions

                //Search Insertions from the left
                int i = startpos;

                //Auxiliary DNA string with the insertion
                //String insrefsequence = (refGene.getSequence().insertBases(startpos,
                    endpos, insBases)).toUpperCase();
                String insrefsequence=(refGene.getSequence().substring(0,startpos+1)+
                    insBases
                                +refGene.getSequence().substring(endpos,refGene.
                                    getSequence().length())).toUpperCase();


                // New Insertions are inserted if the last base of the Insertion is
                    equal
                // to the base prior to the Insertion. After that i and currentIns are
                    move to the left.
                while (insrefsequence.charAt(i) == insrefsequence.charAt(i+insBases.
                    length())){

                        System.out.println(insrefsequence.substring(startpos-10,
                            endpos+10));
                        Insertion newInsertion = new Insertion(i-1,i,insrefsequence.
                            substring(i,i+insBases.length()),this.refGene,null,null);
                        synonymous.add(newInsertion);
                        i--;
                }

                //Search bases from the right of the insertion
                i = endpos;


                // New Insertions are inserted if the first base of the Insertion is
                    equal
                // to the base next to the Insertion. After that i and currentIns are
                    move to the right.
                while (insrefsequence.charAt(i) == insrefsequence.charAt(i+insBases.
                    length())){
                        i++;
                        Insertion newInsertion = new Insertion (i-1,i,insrefsequence.
                            substring(i,i+insBases.length()),this.refGene,null,null);
                        synonymous.add(newInsertion);
                }
                return synonymous;
        }
}
```

Documentation.java, Bibliography.java,

```
/**
* The Documentation class Represents the information required to describe a variation
* that has been discovered by biologists: The associated phenotype and the
    bibliography
* associated and the source where this documentation is located.
*
* @author mariajo
*
*/
public class Documentation {
        public enum Effect{Mutant, SNP}
```

```
        private String phenotype;
        private Bibliography bibliography;
        private String certainty;
        private Effect effect;
        private String source;

        public Documentation(String phenotype,String certainty, Effect effect, String
            source) {
                super();
                this.phenotype = phenotype;
                this.certainty = certainty;
                this.effect=effect;
                this.source=source;
        }

}
/**
* The Bibliography class Represents the information about a publication.
*
* @author mariajo
*
*/

public class Bibliography {
        private String publication;
        private Date date;
        private String authors;
        private String url;
        private String title;

        public Bibliography(String title, String publication, Date date,
                        String authors, String url) {
                super();
                this.title = title;
                this.publication = publication;
                this.date = date;
                this.authors = authors;
                this.url=url;
        }
}
```

org.pros.genoma.diagen.knowledge.xmlReport

Incluye las clases Bibliography.java, change.java, DnaFile.java, DnaVariation.java, Documentation.java, DocumentationList.java, IKnowledgeReportOperations, Phenotype.java, KnowledgeRerpot.java, KnowledgeReportCM.java y VariationList.java.

C.4. Acceso a Datos

org.pros.genoma.diagen.dataAccess

Incluye las clases ENSEMBL.java, HGDB.java, IHGDBDataAccess.java y OracleHGDB.java.

HGDB.java

```
public abstract class GenomaDb implements IGenomaDbDataAccess{
private enum VarType {IS,DE,ID};
protected Connection connection;
public String errorGDb;

public Connection getConnection() {
        return this.connection;
}
public String getError(){
        return this.errorGDb;
}
public void clearError(){
        this.errorGDb=null;
}
/*----------------------------FrameworkQueryOperationsmethods----------*/
public List<Documentation> checkVariationByPos(PreciseVariation var){
        List<Documentation> list=new ArrayList<Documentation>();
        /*Check the variation depending on the type and position*/
        RowSet rs=this.query(stringQueryByPos(var));
        try {
                if(rs.isFirst()){//Any occurrence
                        /*Cover all matches and search each documentation*/
                        for(rs.first();rs.isAfterLast()==false;rs.next()){
                                Documentation d;
                                if((d=this.retrieveDocumentation(rs.getInt("
                                    ID_VARIATION")))==null){
                                        String error="Error retrieving documentation";
                                        System.err.println(error);
                                }
                                list.add(d);
                        }
                        return list;
                }
        } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        return null;//variation not found

}

/*Get list of documentations about a variation searching by flanking sequences*/
public List<Documentation> checkVariationByFlanking(PreciseVariation var,float
    ensibility){
        List<Documentation> list=new ArrayList<Documentation>();

        /*SQL code to query variation by flanking sequences*/
        /*QUery variation by gene, by type and by sequence changed*/
        String pvSelect = stringQuery(var);
        RowSet rs=this.query(pvSelect);
        try{
        /*Check all variations and get those whose flanking sequence are similar
            sensibility)*/
        if(rs.isFirst()){
                for(rs.first();rs.isAfterLast()==false;rs.next()){
                        /*Compare flanking sequences*/
                        int fsEqual=0;
                        /*FLANKING LEFT*/
                        String fsl=rs.getString("FLANKING_LEFT");
                        for(int i=0;i<fsl.length();i++){
                                if(fsl.charAt(i)==var.getFsl().charAt(i)){
                                        fsEqual++;
                                }
```

```
                        }
                        /*FLANKING RIGHT*/
                        String fsr=rs.getString("FLANKING_RIGHT");
                        for(int i=0;i<fsr.length();i++){
                                if(fsr.charAt(i)==var.getFsr().charAt(i)){
                                        fsEqual++;
                                }
                        }
                        /*Search documentation if sensibility matches*/
                        /*    Equal/Total >=sensibility    */
                        if((float)((float)fsEqual/(float)(fsr.length()+fsl.length()))
                            >=sensibility{
                                Documentation d=null;
                                if((d=this.retrieveDocumentation(rs.getInt("
                                    ID_VARIATION")))==null){
                                        String error="Error retrieving documentation
                                            of variation found with {"+
                                                    var.getFsl()+","+var.getFsr
                                                        ()+"}";
                                        System.err.println(error);
                                }
                                list.add(d);
                        }
                }
        }
        }catch(Exception e){
                e.printStackTrace();
        }
        return list;
}
public Reference retrieveReferenceById(String idSymbol){
        /*SQL Code: Gene, Allele and DBank info*/
        String genSelect = "Select * From Allele A, Allelic_reference_type ART,
            Allele_databank_ident DB " + "Where id_gene='" + idSymbol +"' and A.
            ALLELE_NUM = ART.ALLELE_NUM and DB.ALLELE_NUM=A.ALLELE_NUM";
        RowSet rs = this.query(genSelect);
        Reference r=null;
        try {
                if(rs.first()){ //id, RefDB or accessioNumber, sequence
                        r=new Reference(idSymbol);
                        r.setSequence(rs.getString("SEQUENCE"));
                        r.setAccession(rs.getString("ID_ALLELE_DB"));
                }
        } catch (Exception e) {
                e.printStackTrace();
        }
        return r;
        }
public List<PreciseVariation> retrieveVariationsByGene(Gene g) {
        List<PreciseVariation> list = new ArrayList<PreciseVariation>();
        String varSelect = "Select P.ID_VARIATION, POSITION, TYPE, INS_SEQUENCE,
            INS_REPETITION, NUM_BASES, SNP,"+ "FLANKING_LEFT, FLANKING_RIGHT" + " From
            Precise P, Allele A, Variation V"+ " Where A.ID_GENE= '"+ g.getIdSymbol() +
            "' AND A.ALLELE_NUM = V.ID_ALLELE_NUM_RT AND V.ID_VARIATION = P.
            ID_VARIATION";
        RowSet rs = this.query(varSelect);


        try {
         rs.beforeFirst();
         while(rs.next()){
                 PreciseVariation pv = null;
                 int startPos = rs.getInt("POSITION");
                 int numBases = rs.getInt("NUM_BASES");
                 int endPos = rs.getInt("POSITION") + numBases;
```

```
                String bases = rs.getString("INS_SEQUENCE");
                String fsRight = rs.getString ("FLANKING_RIGHT");
                String fsLeft = rs.getString ("FLANKING_LEFT");
                if(rs.getString("TYPE").equals("ID")){
                        pv = new Indel(startPos, endPos-1, bases, g, fsRight, fsLeft)
                            ;
                }
                else if(rs.getString("TYPE").equals("DE")){
                        pv = new Deletion (startPos, endPos, numBases, g, fsRight,
                            fsLeft);
                        ((Deletion) pv).setDelBases(bases);
                }
                else if(rs.getString("TYPE").equals("IS")){
                        pv = new Insertion (startPos, endPos, bases, g, fsRight,
                            fsLeft);
                }
                list.add(pv);
        }
        } catch (SQLException e) {
                e.printStackTrace();
        }
        return list;
        }

public Gene retrieveGeneById (String gSymbol){
        Gene g=null;
        String genSelect = "Select A.ALLELE_NUM, ID_ALLELE_DB, SEQUENCE "+
                "From Allele A, Allele_Databank_Ident D, Allelic_Reference_Type R "+
                "Where A.ALLELE_NUM = D.ALLELE_NUM AND D.ALLELE_NUM = R.ALLELE_NUM
                    AND " +
                "A.ID_GENE='"+gSymbol+"'";

                RowSet rs = this.query(genSelect);
        try {
        String allele_num = rs.getString("ALLELE_NUM");
        String id_allele = rs.getString("ID_ALLELE_DB");
        String sequence = rs.getString("SEQUENCE");
        if(rs.first()){
                g= new Gene(gSymbol,id_allele,sequence,allele_num,null);
        }
        } catch (Exception e) {
                e.printStackTrace();
        }
        return g;
        }
/*----------------------Private methods-----------------------------*/
private String stringQueryByPos(PreciseVariation var){ //Sequence in database case
    sensitive (all in uppercase)
                /*SQL code to query variation by type, by gene and by sequence changed
                    */
                String pvSelect = stringQuery(var);
                /*SQL code to query variation by position also*/
                switch(getType(var)){
                case IS://In case of insertion the position to check is endpos;
                        pvSelect += " and position = " + var.getEndpos() ;
                        break;
                case DE: //In case of deletion the position to check is startpos;
                        pvSelect += " and position = " + var.getStartpos();
                case ID://In case of indel the position to check is startpos;
                        pvSelect += " and position = " + var.getStartpos();
                        break;
                }
                return pvSelect;
        }
```

```
        private String stringQuery(PreciseVariation var){
                /*SQL code to query variation by type, by gene by sequence changed*/
                String pvSelect = "Select * from Precise P, Variation V, allele A";
                switch(getType(var)){
                case IS:
                        pvSelect += " where type = 'IS' and " + "ins_sequence like '"
                            +((Insertion)var).getInsBases().toUpperCase()+"'";
                        break;
                case DE:
                        pvSelect += " where type = 'DE' and " + "num_bases="+((
                            Deletion)var).getDelLength();
                        break;
                case ID:
                        pvSelect += " where type = 'ID' and " + "ins_sequence like '"
                            +((Indel)var).getInsBases().toUpperCase()+"'";
                        break;
                }

                pvSelect+=" and p.id_variation = v.id_variation and" +
                " v.id_allele_num_rt = a.allele_num and a.id_gene='" + var.getRefGene
                    ().getIdSymbol()+"'";
                return pvSelect;
        }

        private VarType getType(PreciseVariation var){
                VarType type=null;
                String name=var.getClass().getName();

                if(name.compareToIgnoreCase(Insertion.class.getName())==0){//Insertion
                        type=VarType.IS;
                }
                else if(name.compareToIgnoreCase(Deletion.class.getName())==0){//
                    Deletion
                        type=VarType.DE;
                }
                else {//if(name.compareToIgnoreCase(Indel.class.getName())==0){//Indel
                        type=VarType.ID;
                }
                return type;
        }
private Documentation retrieveDocumentation(int idVariation){
        Documentation documentation=null;
        try{/*Code to query the effect and db source of variation*/
                String effectSelect= "Select * from Precise P, Variation V where v.
                    id_variation="+idVariation+" and p.id_variation=v.id_variation";
                RowSet rs1=this.query(effectSelect);
                String source="-";
                Documentation.Effect effect=Documentation.Effect.Mutant;//By default
                if(rs1.isFirst()){
                        if(rs1.getString("SNP").equals("1")){//SNP
                                effect=Documentation.Effect.SNP;
                        }
                        source=rs1.getString("ID_DATA_BANK");
                }
                /*Code to query Documentation*/
                String documentationSelect= "Select * from Certainty C, Phenotype P" +
                        " where c.id_variation = " + idVariation + " and" +
                        " c.id_phenotype = p.id_phenotype";
                RowSet rs2=this.query(documentationSelect);
                if(rs2.isFirst()){ //Documentation found
                documentation=new Documentation(rs2.getString("NAME"),rs2.getString("
                    LEVEL_CERTAINTY"),effect,source);
                }else {/*No documentation found: Causes:
                 2)*/if(effect.compareTo(Documentation.Effect.Mutant)==0){// Phenotype
                        is not documented in original source (error)
```

```
                                        documentation=new Documentation("Not
                                            Documented","-",effect,source);
                                        String error="Error retrieving phenotype of a
                                            mutation found ("+ idVariation+")";
                System.err.println(error);
                //return null;
                }
                else{//Variation is a SNP (correct),
                        documentation=new Documentation("SNP","-",effect,source);
                }
        }
        /*Code to query url. Provisional solution*/
        Bibliography biblio=null;
        if(effect.compareTo(Documentation.Effect.Mutant)==0){//Mutation
                /*Code to query Bibliography*/
                String bibliographySelect= "Select * from Reference_Variation RV,
                    Bibliography_Reference BR, Bibliography_DB_Location BDBL" +
                                " where rv.id_variation = " + idVariation  + " and" +
                                " rv.id_bib_ref = br.id_bib_ref"/*+ " and rv.
                                    id_bib_ref = bdbl.id_bib_ref"*/;
                                /*String bibliographySelect= "Select * from
                                    Reference_Variation RV, Bibliography_DB_Location
                                    BDBL" +
                                " where rv.id_variation = " + idVariation  + " and rv.
                                    id_bib_ref = bdbl.id_bib_ref";*/

                RowSet rs3=this.query(bibliographySelect);
                if(rs3.isFirst()){
                        /*Date*/
                        SimpleDateFormat df= new SimpleDateFormat(" yyyy MMM");
                        java.util.Date date=null;
                        try {
                                date = df.parse(rs3.getString("DATE_PUB"));
                        } catch (ParseException e) {
                                e.printStackTrace();
                        }
                        biblio=new Bibliography(rs3.getString("TITLE"),rs3.getString(
                            "PUBLICATION"), date, rs3.getString("AUTHORS"),rs3.
                            getString("URL"));
                                }
                        }else { //SNP
                                String bibliographySelect= "Select id_variation_db
                                    from Variation where id_variation ="+ idVariation;
                                RowSet rs3=this.query(bibliographySelect);
                                if(rs3.isFirst()){
                                        biblio=new Bibliography("rs"+rs3.getString("
                                            ID_VARIATION_DB"),null,null,null,null);
                                }
                        }
                        /*Add bibliography to documentation*/
                        if(biblio!=null){
                                documentation.setBibliography(biblio);
                        }else{
                                String error="Error retrieving bibliography of a
                                    variation found ("+ idVariation+")";
                                System.err.println(error);
                        }
                }catch(SQLException e) {
                                e.printStackTrace();
                }
                return documentation;
        }
        public List<String> getGeneList(){

                /*SQL Code: Gene, Allele and DBank info*/
```

```
                String genListSelect = "Select id_symbol From Gene";
                RowSet rs = this.query(genListSelect);
                /*Create List*/
                List<String> genes=new ArrayList<String>();
                try {
                        for(rs.first();rs.isAfterLast()==false;rs.next()){ //id, RefDB
                             or accessioNumber, sequence
                                genes.add(rs.getString("ID_SYMBOL"));
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                }
                 return genes;
        }
}
```