Universidad de las Ciencias Informáticas

FACULTAD 6

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

"SEDEGEN: Sistema Experto para el Diagnóstico de Enfermedades Genéticas"

Autores: Asiel Leal Celdeiro

Víctor Manuel Pelegrino Ros

Tutores: Ing. Dayana Joseph Smarth

Ing. Yudiel La Rosa González

Consultante: DrC. Roberto Lardoeyt Ferrer

La Habana, Junio 2014

"Año 56 de la Revolución"

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Asiel Leal Celdeiro

_____

Firma del Autor

Víctor Manuel Pelegrino Ros

_____

Firma del Autor

Yudiel La Rosa González

_____

Firma del Tutor

Dayana Joseph Smarth

_____

Firma del Tutor

Datos de contacto

Autores:

Asiel Leal Celdeiro
Universidad de las Ciencias Informáticas, La Habana, Cuba
Email: aceldeiro@estudiantes.uci.cu

Víctor Manuel Pelegrino Ros
Universidad de las Ciencias Informáticas, La Habana, Cuba
Email: vmpelegrino@estudiantes.uci.cu

Tutores:

Ing. Yudiel La Roza González
Universidad de las Ciencias Informáticas, La Habana, Cuba
Email: ylarosag@uci.cu

Ing. Dayana Joseph Smarth
Universidad de las Ciencias Informáticas, La Habana, Cuba
Email: djoseph@uci.cu

Consultante:

DrC. Roberto Lardoeyt Ferrer
Centro Nacional de Genética Médica, La Habana, Cuba
Email: lardgen@infomed.sld.cu

Agradecimientos

A nuestros tutores, por su apoyo y ayuda para que este trabajo quedara con la calidad que amerita, por el tiempo y la preocupación dedicada a nosotros. Nunca tendremos palabras para expresar nuestro agradecimiento por ayudarnos a convertirnos en ingenieros.

A nuestro amigo, el Dr. Roberto Lardoyet Ferrer, por toda la información brindada, por su apoyo y por las horas dedicadas a la realización de este trabajo.

A todos los que nos apoyaron y brindaron sus ideas para poder alcanzar nuestras metas, especialmente a Reisel, quien de cerca siempre se preocupó por nosotros y a Diana por darnos la mano cada vez que la necesitamos.

A todos los profesores y especialistas que han contribuido a nuestra formación como ingenieros.

De Asiel:

A mis dos madres, por todo el sacrificio que han hecho para que hoy finalmente pueda ser un profesional.

A mi papá, por haber sido el ejemplo que necesitaba y darme todo su apoyo.

A mi primo Hectico, por la invaluable ayuda que me ha dado.

A toda mi familia por ser tan unida y brindarme siempre su apoyo, especialmente a mis tíos y tías quienes han sido siempre mis padres, gracias.

De Víctor:

A mi madre por su dedicación, su empeño y sacrificio en mi formación como profesional.

A mi padre por su apoyo y por haber sido un referente en mi formación como persona y profesional.

A mi hermano que es y será mi orgullo y siempre me ha apoyado en los momentos más difíciles.

A mi familia que siempre me ha impulsado a seguir adelante y siempre está en los momentos buenos y malos, mis tías, mis tíos, mis primos, mis abuelos, gracias por su apoyo incondicional.

Quisiera agradecer a quienes han sido también parte de mi familia Damary, Peña , Abel, Giselle, Omaida, José Enriquez, Yaimara, Alberto, Daymara, Dayana por soportarme por tantos años y siempre ayudarme en todo momento.

Por último pero no menos importante a quienes han sido más que amigos hermanos Dawi, Ernesto y Adrián por haber estado a mi lado en las buenas y malas.

Dedicatoria

De Asiel:

Quiero dedicar este trabajo especialmente a mis hermanitas: Marian, Diana, Dania y Yenet, quienes pronto se convertirán también en profesionales.

A mis padres, quienes son mi orgullo y mi inspiración para ser una mejor persona cada día.

A mis abuelas, que no por llevar más de seis años viviendo lejos de casa, dejo de extrañar todos los días.

A todos mis tíos y tías, a mis primos y primas, que más que eso, son hermanos para mí, a toda mi familia: este trabajo también es de ustedes.

De Víctor:

Quisiera dedicar este trabajo especialmente a mi hermano Camilo que es mi razón de ser, quien pronto será también un profesional.

A mis padres Rosa Elena y Víctor Manuel que me trajeron al mundo y me formaron como mejor persona.

A mi primo Marcos por su apoyo incondicional y ayuda en todo momento.

A mis tías Clara y Martha, a mi tío Marco que siempre ha sido como un padre para mí. A mi abuela Dulce y mi prima Dulcita. A toda mi familia le dedico este trabajo.

Resumen

Las enfermedades genéticas son desviaciones del estado de salud debido a la constitución genética de las personas. Cada año aumenta la aparición de nuevas enfermedades de este tipo; esto, unido a la rareza individual de cada una de ellas, dificulta su correcta identificación y posterior tratamiento de los pacientes. Actualmente el diagnóstico de tales enfermedades en Cuba constituye una labor de primer orden para el Centro Nacional de Genética Médica. Debido a las limitaciones que presenta el mismo en cuanto a la consulta de información referente al tema, sus especialistas deben realizar un esfuerzo adicional para tratar a cada paciente. En numerosas ocasiones deben consultar libros, discutir los nuevos casos con otros especialistas y estudiarlos minuciosamente. De esta forma, para emitir un diagnóstico efectivo, muchas veces se requiere una cantidad considerable de tiempo y esfuerzo. Por tal motivo se desarrolló el Sistema Experto para el Diagnóstico de Enfermedades Genéticas: SEDEGEN. El mismo, además de brindarles a los genetistas información acerca de las enfermedades genéticas, es capaz de determinar cuál de ellas es más probable que padezca un paciente, a partir de los síntomas que presente. El sistema experto usa reglas de producción que contienen el conocimiento de un especialista del Centro Nacional de Genética Médica y de la literatura especializada en el tema, realizando un proceso de inferencia con dirección de búsqueda hacia adelante. Las principales tecnologías utilizadas fueron Java, como lenguaje de programación, Drools, como Sistema Gestor de Reglas y SQLite como Sistema de Gestión de Bases de Datos.

Palabras claves

Diagnóstico, enfermedades genéticas, genética, inteligencia artificial, sistema experto

Abstract

Genetic diseases are health state deviations produced by genetic constitution of people. Occurrence of new disease of this kind has been increasing every year; this situation and the individual oddness of each one of them, make difficult their correct identification and subsequent treatment of patients. Nowadays diagnosis of that type of diseases in Cuba constitutes a first order work for the Medical Genetics National Center, but due to its limitations regarding to consulted information about that topic, its specialists have to make an additional effort to treat every patient. In many occasions they have to consult books, to discuss the new cases with other specialists and to study them meticulously. Thus, many times to emit an effective diagnosis, a considerable amount of time and effort are required. For that reason, the Expert System for Diagnosis of Genetic Diseases (SEDEGEN) was developed. It provides to geneticists with information about genetic diseases and it´s capable to determine which one of those diseases is more likely to be suffered by patient, having in account the present symptoms. The expert system uses production rules which contains knowledge form a specialist of the National Center of Medical Genetics and specialized literature about the theme, performing inference process with searching direction toward ahead. The main used technologies were Java as programming language, Drools as Rule Management System and SQLite as Database Management System.

Key words

Artificial Intelligence, diagnosis, expert system, genetics, genetic diseases.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Introducción

A principios del siglo XX se realizaron numerosos avances en el campo de la Genética Humana y fueron descubiertas las actuales bases de la genética general. Con los progresos bioquímicos realizados y los nuevos enfoques sobre técnicas de cultivo de tejidos, en el año 1956, se da una visión renovadora de la medicina. Estos acontecimientos influyen directamente en el campo de la genética, disciplina en la que los especialistas del mundo y particularmente los cubanos dedican un gran esfuerzo. En este período, en el campo de la Genética Humana, se toma un interés especial en el desarrollo del estudio de enfermedades raras o generalmente de baja frecuencia, que se acompañaban de varias generaciones de individuos afectados en las familias, y emerge con un nuevo enfoque en la medicina: la Genética Médica.

La Genética Humana es la ciencia que estudia la variación entre los seres humanos basándose en las diferencias del material hereditario, mientras que la Genética Médica es la ciencia encargada de la aplicación de estos principios en la práctica médica (Ojeda, 2013). En esta última existen diferentes ramas, tales como: la citogenética, la genética molecular y bioquímica, la genética de población, la genética del desarrollo y la genética clínica. Especialmente la genética clínica está orientada a ofrecer consultas a los individuos que asisten preocupados en busca de un diagnóstico médico.

En Cuba los primeros en interesarse por el desarrollo de la genética clínica y los primeros en realizar diagnósticos clínicos fueron los pediatras, pero no fue hasta después del 2003 que se diseñó una estrategia donde se incluía la creación del Centro Nacional de Genética Médica (CNGM). Actualmente este centro es el encargado de coordinar nacionalmente la actividad de asistencia médica en la red de centros y servicios de genética médica, así como orientar, evaluar y controlar la ejecución de los distintos subprogramas del Programa de Diagnóstico y Prevención de Enfermedades Genéticas en el país.

Es importante destacar que "*el principal objetivo de las aplicaciones de la genética a la salud pública es la reducción del impacto de las enfermedades genéticas sobre la salud y el bienestar de los individuos a través de estrategias de prevención. Ello permite ayudar a las personas con desventajas genéticas a vivir y a reproducirse de forma tan normal como sea posible, así como reducir la frecuencia y las manifestaciones clínicas de los defectos congénitos severos*" (Marcheco T, 2008).

Las enfermedades genéticas y los defectos congénitos representan una de las primeras causas de morbimortalidad infantil en la actualidad debido a las manifestaciones clínicas que de manera tórpida pueden afectar la esperanza de vida de las personas que las padecen. El diagnóstico temprano y efectivo de ellas

constituye un factor primordial para trazar estrategias de prevención secundaria y terciaria en el campo de la genética médica y clínica.

En ocasiones resulta difícil identificar algunas de las enfermedades genéticas por su baja frecuencia, muchas veces esto constituye una dificultad al realizar el diagnóstico para los médicos que efectúan sus primeras consultas. Además, cada año es mayor el número de desórdenes genéticos nuevos que se reportan a bases de datos internacionales de gran prestigio, como lo es Herencia Mendeliana en el Hombre en Línea, OMIM (del inglés Online Mendelian Inheritance in Man).

El CNGM no está exento de estos problemas, para tratar a un paciente, los especialistas se basan en su experiencia, en el conocimiento que adquirieron de su formación, se apoyan en la sugerencia de otros expertos y en la literatura especializada. Muchas veces esto resulta ser un proceso engorroso, que provoca el retardo del diagnóstico y desgasta al doctor. Es una necesidad del centro mejorar este proceso, de manera que los especialistas tengan una forma de acceso rápida a la información relacionada con las enfermedades genéticas y puedan consultar una opinión de apoyo para emitir un diagnóstico final.

Por los elementos antes expuestos se define como **problema de la investigación:** ¿Cómo contribuir al diagnóstico médico de las enfermedades genéticas en el CNGM?

Se define como **objeto de estudio:** El desarrollo de sistemas expertos para el diagnóstico de enfermedades.

Según lo planteado anteriormente se define como **campo de acción:** El desarrollo de sistemas expertos para el diagnóstico de enfermedades genéticas.

Para dar solución al problema de la investigación se plantea como **objetivo general:** Desarrollar un sistema experto para el diagnóstico de enfermedades genéticas en Cuba que contribuya a la toma de decisiones de los especialistas del Centro Nacional de Genética Médica.

Para dar cumplimiento al objetivo general, se plantean los siguientes **objetivos específicos**:

- Elaborar el marco teórico de los sistemas expertos usados en el diagnóstico de enfermedades para seleccionar la metodología de desarrollo, las herramientas, y tecnologías a usar en el desarrollo del sistema.
- Realizar el análisis del sistema a desarrollar para identificar los requisitos funcionales.
- Diseñar el sistema experto para definir la organización lógica del mismo.

- Implementar el sistema experto para dar respuesta a los requisitos definidos.
- Probar las funcionalidades del sistema para detectar y corregir posibles errores.

Para llevar a cabo cada uno de estos objetivos se especifican las siguientes **tareas de investigación**:

- Análisis de los sistemas expertos para obtener información de su funcionamiento, mediante la revisión de la documentación del tema de tales sistemas.
- Selección de las herramientas y tecnologías a utilizar en el desarrollo del sistema experto, mediante el análisis de herramientas existentes para la construcción de este tipo de sistemas.
- Especificación de los requisitos funcionales y no funcionales, mediante entrevistas con el cliente, para definir las características que debe poseer el sistema que posibiliten el diagnóstico de enfermedades genéticas.
- Adquisición del conocimiento experto para resolver los problemas del dominio, mediante entrevistas con el especialista del CNGM y la revisión de literatura especializada.
- Definición de la arquitectura del sistema para establecer los componentes que lo conforman y la relación entre ellos, mediante el análisis de la estructura y el funcionamiento de los sistemas expertos.
- Elaboración de los diagramas del diseño para describir la estructura de la implementación, utilizando patrones de diseño.
- Implementación de las funcionalidades del sistema experto para dar respuesta a los requisitos del mismo.
- Diseño de las pruebas a ser aplicadas al sistema, para verificar su correcto funcionamiento.
- Ejecución de las pruebas del sistema experto para detectar posibles no conformidades.

Para guiar el proceso de investigación se plantean las siguientes **preguntas científicas:**

- ¿Cuáles son los fundamentos teóricos asociados al desarrollo de sistemas expertos para el diagnóstico de enfermedades?
- ¿Qué aportes presenta el desarrollo de un sistema para el diagnóstico de enfermedades genéticas en el Centro Nacional de Genética Médica?

Introducción

En el desarrollo del presente trabajo se utilizan los siguientes **métodos de investigación:**

Como método teórico se utiliza el analítico-sintético para realizar el estudio de los sistemas similares al que se desea desarrollar, así como el estudio de las herramientas y metodologías de desarrollo de tales sistemas. Como método empírico se utilizan las entrevistas no estructuradas para identificar qué funcionalidades debe poseer el sistema. Mediante preguntas a los especialistas del CNGM se adquieren distintos puntos de vista de cómo esperan que sea el sistema y como les gustaría que funcionara. Las entrevistas, además, son usadas para obtener opiniones de los especialistas acerca de cómo se deben realizar los diagnósticos de las enfermedades a los pacientes, qué criterios se deben tener en cuenta, y si se debe establecer alguna prioridad entre esos criterios.

El presente documento posee tres capítulos, donde se exponen los resultados de las actividades investigativas y el proceso de desarrollo de la aplicación.

Capítulo 1. Fundamento teórico de la creación de un sistema experto: En este capítulo se presentan varios conceptos referentes a la Inteligencia Artificial, así como de los sistemas expertos y específicamente los utilizados en el área de la medicina para la predicción de enfermedades. Además se exponen ideas fundamentales de las enfermedades genéticas y se describen las herramientas, metodología y técnicas a utilizar para desarrollar el sistema experto.

Capítulo 2. Análisis y diseño del sistema experto: Este capítulo está estructurado de acuerdo a la metodología seleccionada. En cada acápite se abordan los elementos correspondientes a las tres primeras fases de la misma: Planteamiento del Problema, Obtención del Conocimiento Experto y Diseño del Sistema Experto. Además, para guiar el proceso de análisis y diseño, se toman elementos de los principios generales del desarrollo de software, planteados en libros como (Pressman, 2010) y (Sommerville, 2007), aplicables a la mayoría de los proyectos de desarrollo de software.

Capítulo 3. Implementación y prueba del sistema experto: En este capítulo se plasman los principales elementos que se corresponden con las dos últimas fases que se emplean de la metodología: Implementación de un Prototipo y Prueba del Prototipo. Se describen los diagramas de componentes y los principales elementos del estándar de codificación que se emplea. También se muestra cómo se realizó la implementación de la base de conocimiento, así como la forma de expandirla desde el punto de vista de los desarrolladores. Además se presentan los resultados de las pruebas realizadas al software para comprobar la correcta implementación de cada una de las funcionalidades.

Capítulo 1: Fundamento teórico de la creación de un sistema experto

En este capítulo se presentan varios conceptos referentes a la Inteligencia Artificial, así como de los sistemas expertos y específicamente los utilizados en el área de la medicina para la predicción de enfermedades. Además se exponen ideas fundamentales de las enfermedades genéticas y se describen las herramientas, metodología y técnicas a utilizar para desarrollar el sistema experto.

1.1. La Inteligencia Artificial

El término Inteligencia Artificial (IA) es aún muy debatido y no existe un concepto que sea aceptado completamente por la comunidad científica. En algunos libros se plantea que "*la Inteligencia Artificial estudia cómo lograr que las máquinas realicen tareas que, por el momento, son realizadas mejor por los seres humanos*" (Rich, y otros, 2000); en otros, que es el "*esfuerzo por hacer que las computadoras piensen… máquinas con mentes, en el más amplio sentido literal (…) o la automatización de actividades que están asociadas con el pensamiento humano, actividades como la toma de decisiones, resolución de problemas, aprendizaje …*" (Rusell, y otros, 2010). En todos los casos, sin importar el autor o el año en que se planteó la definición, todos coinciden, de una forma u otra, en que la IA se encarga de cómo hacer que las máquinas realicen actividades que requieren de la capacidad de pensar de los seres humanos, o al menos de su conocimiento.

Los problemas abordados por la IA se presentan en muchas aristas de la vida por lo que existen varias técnicas de resolución de los mismos. Los sistemas expertos constituyen una de estas técnicas y resuelven actualmente problemas que son solucionados por "expertos" humanos en algún área del conocimiento. En los últimos años ha habido un notable aumento en la utilización de este tipo de sistemas en muchas esferas de la vida por las particularidades que poseen, permitiéndoles realizar tareas que comúnmente hacen seres humanos.

1.2. Los sistemas expertos

Actualmente los Sistemas Basados en el Conocimiento (SBC) constituyen la mayor parte de la IA aplicada. "*Un SBC es un sistema computarizado que usa conocimiento sobre un dominio para arribar a una solución de un problema de ese dominio. Esta solución es esencialmente la misma que la obtenida por una persona*

experimentada en el dominio del problema cuando se enfrenta al mismo problema" (Gálvez L, 2006). A menudo los términos SBC y sistema experto (SE) se usan de forma indistinta, por lo que en el presente documento se hace referencia solo a SE, sabiendo que se habla del mismo tipo de software. Por tanto se puede afirmar que un SE es un sistema informático (hardware y software) que simula a los expertos humanos en un área de especialización dada.

Para la realización de un sistema de este tipo debe tenerse en cuenta que sea adecuado para el dominio donde se utilizará. Además se debe contar con los recursos de hardware y software necesarios para lograr un rendimiento eficiente. De forma general deben cumplirse los siguientes requisitos:

- La solución de las tareas no requiere sentido común, sólo habilidades cognitivas.
- Existen expertos en el dominio, estos pueden articular sus métodos y se pueden poner de acuerdo.
- Las tareas del dominio están bien comprendidas.

De forma general, la realización de un SE es adecuada cuando la experticia humana puede perderse, escasea, se necesita en muchos lugares o en ambientes hostiles o cuando se requiere mejorar la calidad del conocimiento de los expertos humanos en un área específica.

1.2.1. Estructura de los sistemas expertos

Un elemento importante en la construcción de un SE para que funcione de manera correcta y eficientemente es la organización y colaboración entre sus partes. Existen literaturas que esbozan de diversas formas la estructura de un SE, dependiendo del nivel de abstracción que presenten, pero la mayoría coincide de una forma u otra en los principales elementos (Castillo, y otros, 1996). A continuación se explican estos componentes haciendo uso de la Figura 1, donde la dirección de las flechas representa la dirección del flujo de la información.

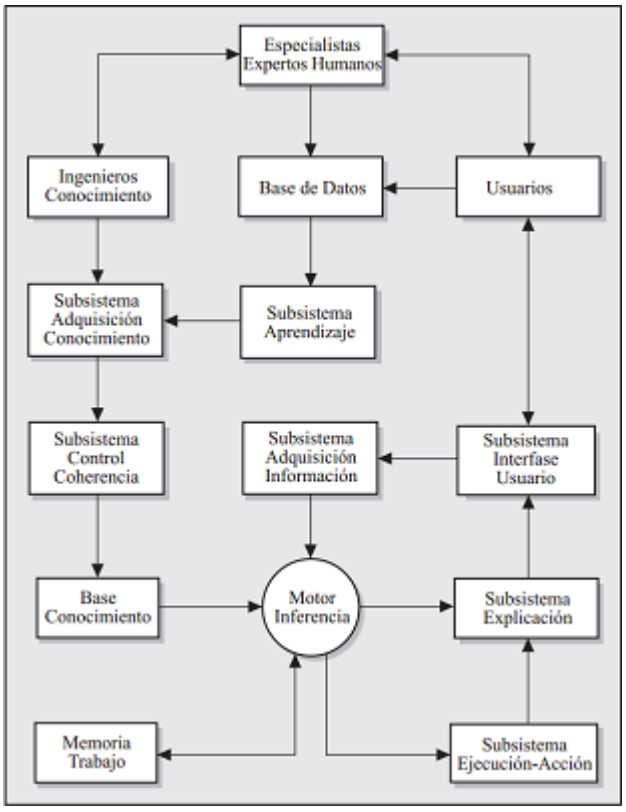


Figura 1. Componentes de un SE.

- Componente humana: A través de la colaboración de Especialistas Expertos Humanos en un área de especialización determinada y los Ingenieros del Conocimiento es que se logra construir un SE de calidad.
- Ingenieros del conocimiento: Son los encargados de obtener el conocimiento experto en la fase de creación del sistema para incorporarlo a este, y se realiza mediante las técnicas de ingeniería del conocimiento.
- Base de conocimiento (BC): Constituida por el conocimiento que brindan los expertos humanos y que transforman los ingenieros del conocimiento para que el sistema computacional lo interprete. Este conocimiento debe estar ordenado y bien estructurado para lograr buenos resultados en las respuestas brindadas por el sistema. Para la conformación de la BC se usa alguna forma de representación del conocimiento (FRC), estas pueden ser, marcos, guiones, redes semánticas y reglas de producción.

- Subsistema de adquisición de conocimiento: Controla el flujo del nuevo conocimiento que fluye del experto humano a la base de datos. Es usado por el sistema para adquirir nuevo conocimiento que no se tenía almacenado.
- Subsistema de control de coherencia: Controla la consistencia de la base de datos y evita que unidades de conocimiento inconsistentes entren en la misma.
- El motor de inferencia (MI): Es el núcleo de todo SE. El principal objetivo de este elemento es obtener conclusiones a partir de los datos del problema (en la memoria de trabajo), analizando el conocimiento en la BC.
- Interfaz de usuario: Es el mecanismo usado para mostrar al usuario los resultados obtenidos por el motor de inferencia y otros subsistemas como el de explicación. De nada vale un buen mecanismo de inferencia si la interfaz de usuario es pobre y no ofrece orientación en sus preguntas y resultados.
- Subsistema de adquisición de información: Es usado por el motor de inferencia para adquirir el conocimiento necesario cuando inicialmente este es muy limitado y no se pueden obtener conclusiones. A partir de aquí se continúa con el proceso de inferencia hasta que se hayan arribado a conclusiones.
- Subsistema de ejecución de órdenes: Permite al SE ejecutar acciones basadas en las conclusiones que se obtienen del proceso de inferencia.
- Subsistema de explicación: Explica el proceso seguido por el motor de inferencia y/o por el subsistema de ejecución de órdenes. Esta explicación se le muestra al usuario a través de la interfaz de usuario.
- Subsistema de aprendizaje: Permite incorporar al SE nuevo conocimiento. Esta es una característica de este tipo de software, que brindan la posibilidad de “aprender”, incorporando nuevo conocimiento a partir de los expertos, aunque esto puede o no suceder, en dependencia del objetivo que se persiga y de la implantación que se realice del sistema.
- Memoria de trabajo: Está formada por los hechos particulares de una situación que debe resolver el SE a partir del conocimiento que posee almacenado en la BC.

1.2.2. Fortalezas de los sistemas expertos

En comparación con otras soluciones, tales como software convencionales, los sistemas expertos tienen varias ventajas, entre las que se encuentra la amplia distribución de experticia escasa, pues cuando existen pocos expertos, aunque poseen el conocimiento de un área determinada, muchas veces no pueden resolver todos los problemas por cuestiones de disponibilidad o de tiempo; de esta forma al transferir ese

conocimiento a un sistema este puede ser utilizado en todos los problemas del área de especialización, incluso al mismo tiempo.

También son fáciles de modificar, pues su conocimiento es explícito y accesible, y una vez creada la BC, el equipo de desarrollo (o cualquier otro con conocimiento en el tema) puede darle soporte y modificar, adicionar o eliminar conocimiento almacenado. Otra importante característica es que muestran consistencia en las respuestas a diferencia, muchas veces, de los expertos humanos, que pueden diferir en sus explicaciones. También facilitan la accesibilidad, ya que trabajan las 24 horas todos los días, se preserva la experticia en la BC y brindan solución a problemas que incluyen datos incompletos.

Estos sistemas ofrecen la explicación de las soluciones, pues justifican sus conclusiones y explican por qué hacen determinada pregunta, mediante el subsistema de explicación. Por último constituyen entrenadores en el dominio de aplicación, pues pueden ser usados con fines educativos para enseñar a especialistas humanos que no dominan a la perfección el conocimiento de ese dominio.

Existen variadas clasificaciones de estos sistemas; estas están dadas por la FRC que utilizan para almacenar la información y cómo se realiza el proceso de inferencia. Entre ellas se encuentran los Sistemas Basados en Marcos, los Sistemas Basados en Redes Semánticas, los Sistemas Basados en Casos y los Sistemas Basados en Reglas, siendo este último grupo uno de los más usados por las facilidades que brindan al permitir crear reglas en un lenguaje muy parecido al lenguaje del dominio de su aplicación.

1.2.3. Sistemas Basados en Reglas

Una de las FRC más utilizadas en los sistemas expertos es un conjunto de reglas de producción de la forma:

SI <antecedentes/condiciones> ENTONCES <consecuentes/acciones>

Los sistemas que usan esta FRC son llamados Sistemas Basados en Reglas (SBR) y son actualmente los más conocidos. Según (Gálvez L, 2006) los SBR son SBC en los que la forma de representación del conocimiento usada son las reglas de producción y como método de inferencia utiliza la regla de modus ponens.

En los SBR el proceso de solución de problemas es crear una cadena de inferencias que constituye un camino entre la definición del problema y su solución. Esta estrategia puede utilizarse cuando las premisas verdaderas de algunas reglas coinciden con las conclusiones de otras y se puede ejecutar de dos formas: hacia adelante o hacia atrás.

El encadenamiento hacia adelante está orientado a encontrar la solución a un problema a partir de los datos disponibles, emparejando los hechos en la memoria de trabajo (condiciones del problema actual) con las condiciones de las reglas, generando así nuevas conclusiones que se añaden a la memoria de trabajo, así sucesivamente hasta alcanzar una meta satisfactoria o hasta que no haya más reglas que ejecutar. Esta dirección de búsqueda también se conoce como orientada a datos. En el encadenamiento hacia atrás se plantean posibles metas (objetivos) que resuelvan un problema dado y se trata de demostrar que son una solución satisfaciendo las condiciones que las producen con base al conocimiento e información disponible, así hasta llegar al estado inicial del problema o no queden reglas por verificar. Esta dirección de búsqueda es conocida también como razonamiento dirigido a objetivos (o metas).

El MI utiliza encadenamiento hacia adelante cuando a partir de los datos conocidos se desea progresar hacia una conclusión, muy útil cuando los datos iniciales son pocos y/o existen muchas posibles soluciones. La estrategia de búsqueda encadenamiento hacia atrás se usa cuando se selecciona una posible conclusión y se trata de probar su validez buscando evidencias que las soporten. Esta estrategia es útil cuando existen muchos datos disponibles de partida, de los que solo una pequeña parte son relevantes.

1.2.4. Aplicaciones de los sistemas expertos

De forma general estos software se pueden aplicar donde hay un problema cuya resolución requiere gran cantidad de conocimiento sobre un área específica que generalmente poseen los expertos humanos. Durante los últimos años ha habido un gran desarrollo de los sistemas expertos por lo que su aplicación es muy basta en varios campos de la sociedad: las finanzas, la industria, la electrónica, el campo militar, la aeronáutica, la agricultura, la arqueología, la geología, la meteorología, la química y la medicina.

Precisamente en la medicina ha tenido un gran auge el uso de estos sistemas debido a la rapidez con que brindan sus soluciones y al nivel de confiabilidad que representan, dependiendo, claro, del conocimiento que se le haya incorporado. La Tabla 1 muestra ejemplos de sistemas expertos con aplicación en alguna rama de la medicina (Peña A, 2006). Particularmente en Cuba en los últimos años, aparejado al avance tecnológico, ha habido un mayor desarrollo de este tipo de sistemas en el área de la medicina, como se muestra en la Tabla 2.

Capítulo 1

Tabla 1. Sistemas expertos desarrollados en el mundo usados en la medicina.

Nombre del sistema	**Descripción**
Casnet	Diagnóstico y tratamiento del glaucoma.
Neurologist	Aplicado en la neurología
Mycin	Diagnóstico y terapia de enfermedades infecciosas bacterianas.
Neomycin.	Basado en Mycin, para la enseñanza en la medicina.
Caduceus	Herramienta de diagnóstico para medicina interna

Tabla 2. Sistemas expertos desarrollados en Cuba usados en la medicina.

Nombre del sistema	**Descripción**
SEDIM-SV (Ferrer M, y otros, 2008)	SE para el diagnóstico de sepsis vaginales. Desarrollado en la Universidad de las Ciencias Informáticas (UCI) en el año 2008.
SEGEDIS (Gutiérrez R, y otros, 2010)	SE para el diagnóstico de enfermedades genéticas con dismorfias. Desarrollado en la UCI en el año 2010.
Sistema experto basado en casos para el diagnóstico de la hipertensión arterial (Cuadrado R, y otros, 2011)	SE para el diagnóstico de la hipertensión arterial. Desarrollado en la ciudad de Santa Clara en el año 2011.
Sistema experto para el tratamiento médico de pacientes con Hipertensión Arterial y Diabetes Mellitus (Cabrera B, y otros, 2012)	SE para la gestión de la información para el diagnóstico de la Hipertensión Arterial y Diabetes Mellitus en el Módulo para la Toma de Decisiones del Sistema Integral para la Atención Primaria de Salud. Desarrollado en la UCI en el año 2012
Sistema experto basado en reglas para el apoyo al diagnóstico de la fragilidad en el adulto mayor (Serrano S, y otros, 2013)	SE para el apoyo al diagnóstico de la fragilidad en el adulto mayor. Desarrollado en la UCI en el año 2013.

1.3. Los sistemas expertos en la Genética Médica

Las enfermedades genéticas son las desviaciones del estado de salud debido a la constitución genética del individuo y es producido por anomalías genéticas. Estas anomalías pueden presentarse de variadas formas y pueden estar solas o acompañadas de otras alteraciones, en cuyos casos la asociación puede tomar un patrón específico recurrente o puede ser al azar. Esto es reafirmado por el planteamiento de Jones en (Jones, 2007), cuando afirma que *"(…) todo defecto estructural* [del cuerpo humano] *constituye un error morfogenético congénito (…)"*.

Según (Tamayo F, y otros, 1996) la Dismorfología hace referencia al estudio de tales defectos congénitos, es decir, alteraciones corporales que se originan antes del nacimiento. Las anomalías dismórficas pueden ocurrir en cualquier parte del cuerpo y la mayoría se originan en el primer trimestre del embarazo. Desde el punto de vista de la patología del desarrollo, estos tipos de patrones de defectos estructurales (conocidos como secuencias), pueden dividirse en cuatro categorías (Figura 2).

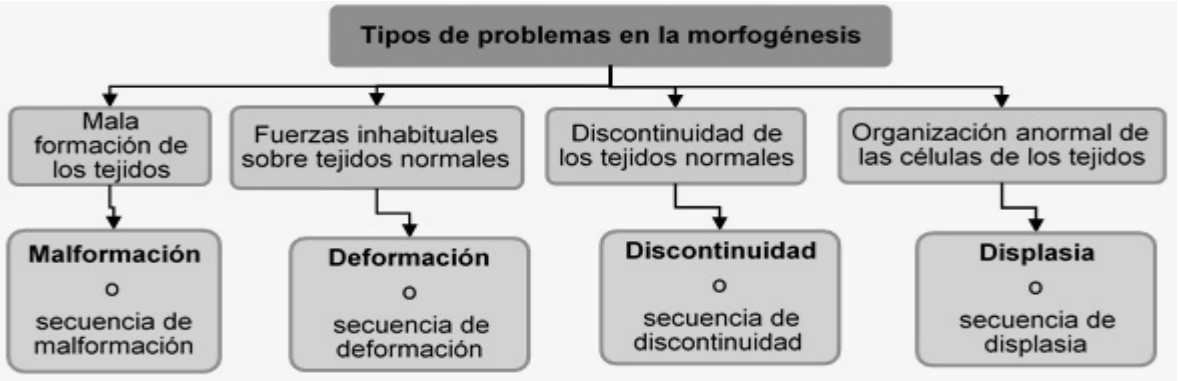


Figura 2. Tipos de defectos estructurales que pueden causar una cadena de defectos al nacer.

La primera categoría es la secuencia de malformación, en la que existe una sola malformación localizada de tejido que inicia una cadena de posteriores defectos. La segunda es la secuencia de deformación, en la que no existe problema en el embrión (feto) pero un trastorno morfogenético es causado por fuerzas mecánicas. La tercera categoría es la secuencia de discontinuidad (o disrupción), en la que el feto normal está sometido a un trastorno destructivo y sus consecuencias. Este tipo de disrupciones pueden tener un origen infeccioso, vascular o mecánico. Por último, la secuencia de displasia, donde el defecto primario reside en la falta de organización de las células de uno o varios tejidos. La mayor parte de los pacientes con

múltiples defectos estructurales se halla en una de las categorías mostradas en la Figura 3. El pronóstico, el manejo y el asesoramiento sobre riesgo de ocurrencia pueden variar mucho de una categoría a otra.

Figura 3. Categorías de los defectos estructurales según la naturaleza del problema.

En el tratamiento de los recién nacidos con múltiples malformaciones, el diagnóstico preciso de un síndrome específico constituye un prerrequisito necesario para el plan de tratamiento del niño y el asesoramiento genético de los padres. Para realizar un correcto diagnóstico se debe tener en cuenta la génesis (bases genéticas), que no es más que los orígenes, la procedencia, características generales que se ha comprobado que presenta la enfermedad, y en la mayoría de los casos ha sido abordado por varios autores. Otro elemento importante son las características[1], es decir, rasgos que identifican la enfermedad y pueden ser signos y/o síntomas. Los signos son aquellos elementos que el especialista puede apreciar en el paciente, es decir características físicas, observables, por ejemplo pelo fino y flexible, pene relativamente pequeño y amplio espacio interdigital en los dedos de los pies. Los síntomas son aquellas características que a veces no son observables pero el paciente puede especificarlos, como dolores y sensaciones.

El manejo y consejería son elementos cruciales en el trato con enfermedades genéticas, pues definen el procedimiento a seguir para establecer un correcto seguimiento del paciente así como brindar un dictamen acertado del tipo de enfermedad y a partir de este, trazar el tratamiento más adecuado. Un último elemento a tener en cuenta para realizar el diagnóstico de las enfermedades genéticas es la diagnosis diferencial,

[1] Generalmente cada enfermedad provoca ciertas características en todos los pacientes, estas son abordadas en el documento como *características obligatorias*. Es decir, un paciente que padece la enfermedad X, casi seguro presenta las características obligatorias de la enfermedad X (al menos una de ellas). El resto de las características son abordadas como *características ocasionales* (o no obligatorias) y se presentan con menor frecuencia en los pacientes.

mediante la cual se puede identificar la enfermedad, excluyendo otras posibles causas que presenten un cuadro clínico semejante al del paciente.

Actualmente existen en el mundo sistemas para el diagnóstico de enfermedades genéticas, pero son liberados bajo licencias privativas y a un alto costo que muchas instituciones como el CNGM no pueden pagar. En la Tabla 3 se muestran algunos de estos sistemas con sus respectivos precios (RAmEx Ars Medica, 2002). En algunos casos se brindan actualizaciones de descarga gratuita para tales sistemas, pero solo aquellos que hayan comprado el software pueden usarlas.

Tabla 3. Aplicaciones existentes en el mundo para el diagnóstico de enfermedades genéticas.

Nombre	**Licencia**	**Precio (Dólares)**
Winter - Baraitser Dysmorphology Database (London DD), 1.0.26, CD-ROM para un usuario	Copyright	$ 1,320.00
Dysmorphology & Neurogenetics Databases & Photo Library, 1.0.22, CD-ROM para un usuario	Copyright	$ 2,256.00
Winter - Baraitser Dysmorphology Database (London DD), 3.0, Licencia de Red, CD-ROM	Copyright	$ 3,960.00
Dysmorphology & Neurogenetics Databases & Photo Library, 1.0.22, Licencia de Red, CD-ROM	Copyright	$ 6, 768.00

Otra forma de obtener información acerca de las enfermedades genéticas es consultando la base de datos OMIM. Esta prestigiosa herramienta está destinada a ser usada principalmente por médicos y profesionales interesados en los trastornos genéticos. No obstante, a ella puede acceder cualquier usuario que tenga internet a su alcance, pero para realizar un diagnóstico preciso se requiere de la consulta de un especialista en el tema. El uso de la OMIM presenta dos inconvenientes principales para el CNGM, primero que la conexión a internet es muy lenta y limitada, y segundo, que esta opción es para hacer búsquedas de información concerniente a alguna enfermedad, no para realizar diagnósticos. Los especialistas solo consultarían información y a partir de los datos obtenidos usarían sus habilidades para diagnosticar al paciente.

Los sistemas expertos han demostrado ser muy eficientes en áreas donde se requiere procesar grandes volúmenes de información para tomar decisiones y hacerlo de una forma relativamente rápida. Cuando

existe información que abarque todo el conocimiento que se tiene hasta el momento del dominio de aplicación y los requisitos de hardware así lo permiten se puede construir una aplicación de este tipo para la toma de decisiones.

En el caso del CNGM, se cuenta con la información referente a las enfermedades genéticas, dígase, características, órganos y sistemas de órganos que afecta, bases genéticas y bibliografía. Esta información está almacenada en libros, además de contar con especialistas que pueden corroborarla y añadir información adicional sobre enfermedades descubiertas después de la publicación de dichos libros, y de hacerle alguna posible corrección a la literatura de acuerdo a los últimos descubrimientos en este campo. Esto unido a la necesidad de acceder de una forma rápida y eficiente a tal información condujo a la creación del Sistema Experto para el Diagnóstico de Enfermedades Genéticas: SEDEGEN.

A través de la información existente de las enfermedades genéticas se puede obtener el conocimiento necesario para realizar el proceso de inferencia con el objetivo de obtener un probable diagnóstico. De esta forma se pueden crear reglas de producción como FRC, donde los antecedentes están constituidos por las características y las estructuras afectadas que presente como cuadro clínico la enfermedad y los consecuentes son los nombres de las enfermedades. De esta forma el SE que se construye es un Sistema Experto Basado en Reglas.

1.4. Metodologías de desarrollo de sistemas expertos

Según la Real Academia Española la palabra metodología se define de la siguiente forma:

- Ciencia del método
- Conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal.

Al relacionar estos conceptos con la Ingeniería de Software, algunos definen metodología como un conjunto de actividades para llevar a cabo la construcción de un software, otros, como la disciplina que estudia los métodos para hacer software. Así se pueden encontrar ideas que tienen puntos de contacto, pero que además difieren en algún sentido. Por tal motivo en el presente trabajo se asume como concepto de metodología el planteado por (Henao C, 2001) en la tesis doctoral para la creación de la metodología CommonKADS-RT mostrado a continuación:

"*Una metodología es un conjunto de métodos, prácticas, estilos, recursos y conocimientos que permiten desarrollar de manera efectiva y eficiente cada una de las actividades que son necesarias para analizar,*

diseñar, producir, implantar y mantener un artefacto". En este caso el concepto de artefacto se refiere a cualquier documento o software que se produzca.

El área de desarrollo de los sistemas expertos es relativamente reciente en comparación con el desarrollo de software convencional por lo que existen muchas metodologías propuestas por desarrolladores de este tipo de sistemas y adaptadas a sus necesidades. Es por ello que no existe una estándar, que unifique todas las ideas planteadas por estos desarrolladores, sino que las que más éxito han tenido se han divulgado y mejorado en el mundo de desarrollo de estos sistemas. Entre esas metodologías se pueden mencionar la metodología CommonKADS, la metodología de Buchanan y la propuesta por Weiss y Kulikowski. En el presente documento se explica brevemente en qué consiste cada una de ellas y se realiza una comparación para determinar cuál resulta más factible aplicar en el desarrollo del sistema.

1.4.1. Metodología de desarrollo CommonKADS

Esta metodología fue desarrollada en la Universidad de Ámsterdam primeramente como un método con el nombre KADS (del inglés Knowledge Acquisition Design System). Por los buenos resultados obtenidos se amplió el proyecto a la construcción de una metodología completa para el desarrollo de sistemas expertos, la cual empieza desde el análisis mismo de la organización en donde se va a hacer el sistema hasta la gestión del proyecto, pasando por el diseño del software, es entonces propuesto y aceptado el nombre de CommonKADS.

La metodología está fundamentada en el modelo del ciclo de vida en espiral y está formada por una serie de etapas, cada una con tareas y productos asociados (Jova R, 2012). Estas son:

- Análisis: Está enfocado a comprender el problema desde el punto de vista de la solución que se piensa desarrollar y se realiza la especificación de requisitos. En esta etapa se deben obtener un documento del proyecto, un documento de los requisitos, un documento del modelo conceptual, un documento de viabilidad y un documento de apoyo.
- Diseño: Se hace una descripción del conocimiento del sistema (descripción funcional) y una descripción física en la que se especifica detalladamente cada uno de sus componentes. En esta etapa se debe obtener toda la especificación modular del sistema y la descripción detallada de cómo debe ser el mismo.
- Implantación del sistema: Se considera tanto la integración del software desarrollado como su adaptación en la organización.

- Instalación: Se pone en marcha el sistema con el fin de que comience a operar en la empresa, iniciándose su proceso productivo.
- Uso: Se plantean actividades relacionadas con el manejo mismo del sistema y de las salidas o resultados que este proporciona.
- Mantenimiento y refinamiento del conocimiento.

CommonKADS ofrece un marco para la especificación del conocimiento independiente de la implementación, combinando un conjunto de modelos de conocimiento reutilizable para tareas que se realizan frecuentemente, como por ejemplo el diagnóstico o la planificación. Además, propone un ciclo de vida en donde se indican las fases, las actividades y los productos más relevantes para un proyecto de desarrollo de un SE.

1.4.2. Metodología de desarrollo Buchanan

En esta metodología el desarrollo de un SE es un proceso de adquisición de conocimiento, visto este como un proceso de transferencia y transformación de la experiencia en la resolución de problemas de un dominio específico a un programa de computadora (Palma, y otros, 2000). Para esto se proponen las siguientes etapas:

- Identificación: Se reconocen aspectos importantes del problema (características, participantes, recursos).
- Conceptualización: Se organiza el conocimiento mediante esquemas conceptuales y se identifica el flujo de información durante el proceso de resolución de problemas.
- Formalización: Se traducen conceptos, subproblemas y características del flujo de información y se construyen representaciones formales basadas en herramientas de desarrollo y esquemas de ingeniería del conocimiento.
- Implementación: Se formulan las reglas, las estructuras de control y se obtiene un prototipo.
- Validación: Se evalúa el rendimiento del prototipo obtenido y se identifican los errores.

El proceso de refinamiento ocurre como causa de la repetición del ciclo en sus dos últimas etapas para de esta forma ajustar la BC y las estructuras de control hasta alcanzar el comportamiento deseado.

1.4.3. Metodología de desarrollo propuesta por Weiss y Kulikowski

Esta metodología, propuesta por Weiss y Kulikowski y descrita por (Castillo, y otros, 1996), sugiere las etapas mostradas en la Figura 4 para el diseño e implementación de un SE:

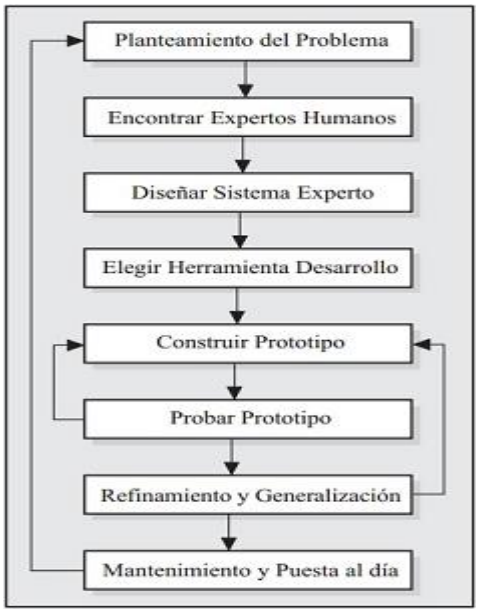


Figura 4. Etapas en el desarrollo de un SE usando la metodología de Weiss y Kulikowski.

- Planteamiento del problema: La primera etapa es la definición del problema a resolver. Debido a que el objetivo principal de un SE es responder a preguntas y resolver problemas, esta etapa es una de las más importantes en el desarrollo de un SE.
- Encontrar expertos humanos que puedan resolver el problema: Los expertos humanos son quienes poseen todo el conocimiento que se incorporará al SE. En algunos casos, sin embargo, las bases de datos o la literatura especializada pueden jugar el papel del experto humano.
- Diseño de un SE: Esta etapa incluye el diseño de estructuras para almacenar el conocimiento, el motor de inferencia, el subsistema de explicación, interfaces de usuario.
- Elección de la herramienta de desarrollo, marco de trabajo, o lenguaje de programación: Debe decidirse si utilizar una herramienta, un lenguaje de programación o un marco de trabajo. Estos últimos son herramientas comerciales que están sujetas a controles de calidad, a los que otros programas no lo están, por lo que generalmente es la opción recomendada, brindando mayor fiabilidad al sistema.

- Desarrollo y prueba de un prototipo: Si el prototipo no pasa las pruebas requeridas, las etapas anteriores (con las modificaciones apropiadas) deben ser repetidas hasta que se obtenga un prototipo satisfactorio.
- Refinamiento y generalización: En esta etapa se corrigen los fallos y se incluyen nuevas posibilidades no incorporadas en el diseño inicial.
- Mantenimiento y puesta al día: En esta etapa el usuario plantea problemas o defectos del prototipo, se corrigen errores, se actualiza el producto con nuevos avances.

Cada una de estas etapas influye en la calidad del software y en el nivel de aceptación del producto final por parte del cliente.

1.4.4. Selección de la metodología de desarrollo de sistemas expertos

Cada una de estas metodologías teóricamente está diseñada para ejecutar de manera exitosa la creación de un SE. La calidad del producto obtenido depende de cuán bien se apliquen en la práctica los aspectos que en ellas se exponen. Por tal motivo en la selección de una de ellas debe tenerse en cuenta las características propias del sistema a desarrollar. En la Tabla 4 se muestra una comparación entre las metodologías mencionadas en cuanto a algunos aspectos importantes que se deben valorar, atendiendo a las características del equipo de desarrollo y del proyecto.

Tabla 4. Comparación entre las metodologías de desarrollo estudiadas.

Metodología	**Documentación generada**	**Incluye la obtención del conocimiento experto**	**Tipo de equipo de desarrollo recomendado (en cuanto a cantidad de desarrolladores)**
CommonKADS	Exhaustiva, todo el proceso genera varios artefactos por cada etapa y fase.	Si	Grande.
Buchanan	Mucha.	Si	Mediano o grande.
Weiss y Kulikowsky	Poca, documentación en dependencia de los intereses del equipo de desarrollo.	Si	Pequeño, aunque no necesariamente.

La metodología CommonKADS tiene como inconveniente que para aplicarla correctamente requiere de mucha experiencia y conocimiento de la metodología misma ya que es muy compleja y amplia, hay mucha información relevante que está en diversos lugares lo que dificulta su acceso y comprensión, no existe una fuente de información que contenga todo lo necesario para su aplicación y no hay un ejemplo completo de la aplicación de la metodología que pueda ser usado como guía. De forma general esta metodología y la Buchanan rigen el proceso de construcción del sistema por una exhaustiva documentación, ya que están pensadas para la construcción de grandes SE empresariales, reflejando incluso interacciones entre múltiples agentes, lo cual no se aplica al contexto del desarrollo del sistema que se aborda en el presente trabajo. La elaboración correcta de toda esa documentación y la aplicación adecuada de cada fase requiere de un personal más numeroso del que se dispone para la construcción del sistema. Estas son las principales razones por la que no se escoge ninguna de las dos para el desarrollo del SE que aborda el presente trabajo.

La metodología propuesta por Weiss y Kulikowski establece las fases, con sus respectivas tareas, para guiar efectivamente el proceso de construcción de un SE sin ser imprescindible la excesiva documentación que otras metodologías exigen. Esta abarca el ciclo de vida del software completamente, desde la concepción del problema a resolver hasta el mantenimiento y soporte. Los desarrolladores que usan esta metodología no necesitan realizar un estudio intensivo de la misma, pues en ella se plantea de manera sencilla y concreta las actividades a realizar. Por tales motivos se escoge la metodología propuesta por Weiss y Kulikowski para la construcción del SE.

Por último es necesario destacar que, si bien la metodología seleccionada abarca desde el planteamiento del problema hasta el mantenimiento y puesta al día del sistema, por las características propias de la presente investigación solo se abarcará hasta la fase de prueba del prototipo. Para poder realizar las dos fases posteriores es necesario implantar el sistema en la organización donde se utilizará, lo que requiere de un tiempo mayor del disponible para el desarrollo del presente trabajo. Quedan definidas, entonces, las siguientes fases para el desarrollo del sistema: Planteamiento del Problema, Obtención del Conocimiento Experto (en la metodología se nombra *Encontrar Expertos Humanos*), Diseño del Sistema Experto, Construcción del Prototipo y Prueba del Prototipo.

1.5. Lenguaje de modelado: UML 2.0

El Lenguaje Unificado de Modelado, UML (del inglés Unified Modeling Language) es un lenguaje de modelado visual que se usa para visualizar, especificar, construir y documentar artefactos de un sistema de

software. Además es usado para la captura de decisiones y conocimiento sobre los sistemas que se deben construir y se usa para entender, diseñar, configurar, mantener y controlar la información sobre tales sistemas (Rumbaugh, y otros, 1998).

UML brinda la posibilidad de modelar las estructuras estáticas y dinámicas de un sistema, a través de las construcciones organizativas para agrupar los modelos en paquetes, permitiendo dividir grandes sistemas en piezas más manejables de trabajo. Este lenguaje da apoyo a la mayoría de los procesos orientados a objetos y está pensado para ser usado en herramientas interactivas de modelado visual, que tengan generadores de informes y de código, como por ejemplo, Visual Paradigm. UML es utilizado en la fase de análisis y diseño del SE para realizar modelos conceptuales, diagramas de paquetes, diagramas de clases, diseño de interfaces y modelos de diseño de la base de datos.

1.6. Herramientas y tecnologías

Una vez definida la metodología de desarrollo, es necesario determinar las herramientas a emplear en la ejecución de cada una de las tareas propuestas en la fase de desarrollo del sistema. Para ello es necesario especificar el lenguaje y la herramienta de modelado, el entorno integrado de desarrollo, IDE (del inglés Integrated Development Environment), las herramientas o los marcos de trabajo para el desarrollo de SE, lenguaje de programación y sistema gestor de base de datos.

1.6.1. Herramienta CASE de modelado: Visual Paradigm 8.0

Visual Paradigm es una herramienta CASE (del inglés Computer Aided Software Engineering) que utiliza UML como lenguaje de modelado bajo el paradigma Programación Orientada a Objetos (POO), para la ayuda en el proceso de desarrollo de software. Brinda confiabilidad y estabilidad en el desarrollo orientado a objetos a ingenieros de software, analistas y arquitectos que están interesados en la construcción de sistemas a gran escala. Ofrece una navegación intuitiva entre la escritura de código y su visualización, un potente generador de informes en formato PDF y HTML y cuenta con un sofisticado diagramador automático de capas.

La versión 8.0 soporta el estándar UML en su versión 2.0 y es compatible con equipos de desarrollo de software en la captura de requisitos, software de planificación (análisis de casos de uso), ingeniería de código, modelado de clase y el modelado de datos. (Visual Paradigm, 2013).

1.6.2. Herramientas existentes que ofrecen motores de inferencia.

Actualmente existen varios motores de inferencia que se pueden usar para el desarrollo de SBR. Independientemente de los elementos comunes entre ellos, cada uno tiene sus particularidades y en función de esto se debe escoger el que mejor responda a las necesidades y objetivos del sistema que se desea desarrollar. Los más mencionados en la web y usados en el mundo comercial son los mostrados a continuación.

- JESS (del inglés Java Expert System Shell).

JESS es un motor de reglas y un entorno de programación escrito enteramente en Java. Puede ser utilizado en la construcción de SE como una máquina de reglas o como un lenguaje de programación. Este usa una forma mejorada del algoritmo Rete[2] para asociar las reglas con la base de conocimiento y permite además el encadenamiento hacia adelante emparejando hechos en la memoria de trabajo con antecedentes y el encadenamiento hacia atrás emparejando hipótesis en la memoria de trabajo con consecuentes. Además puede manipular y razonar sobre objetos java directamente. De acuerdo con (Friedman-Hill, 2013) JESS está disponible sin costo alguno para uso de universidades y laboratorios de Estados Unidos con fines investigativos y/o docentes, y puede ser liberado, además, bajo licencia para uso comercial, por la cual habría que pagar. El uso de esta herramienta en la construcción de software liberados bajo licencia de código libre (*open source*) está prohibido.

- JEOPS (del inglés Java Embedded Object Production System)

Este es un motor de inferencia para Java/JavaScript basado en el encadenamiento hacia adelante, usado para fortalecer los procesos de negocio basados en reglas en aplicaciones servidoras de Java, aplicaciones clientes y Servlets, actualmente bajo la licencia LGPLv2 (del inglés Lesser General Public License) (SourceForge, 2013). JEOPS funciona como un compilador que traduce las reglas de producción escritas en un documento de texto a clases Java, las cuales implementan el motor de inferencia manejado por estas reglas. A partir de aquí el proceso de inferencia se realiza a través del algoritmo Rete mediante el proceso de emparejamiento, además de algunas estrategias de establecimiento de prioridad de reglas.

[2] Algoritmo Rete: Algoritmo creado por Dr. Charles Forgy que describe como las reglas son procesadas en la memoria de trabajo para generar una red de discriminación eficiente.

- CLIPS (del inglés C Language Integrated Production System)

CLIPS fue creado por el Centro Espacial Johnson de la NASA basándose en el lenguaje LIPS (del inglés Language Integrated Production System), pero usando el lenguaje de programación convencional C y usando el algoritmo Rete en el proceso de encadenamiento hacia adelante. Se puede usar sin ninguna restricción: copia, modificación, mezcla, publicación, distribución y/o venta de copias del software. CLIPS provee un entorno completo para la construcción de reglas y/o sistemas expertos basados en objetos. A través del sitio oficial del proyecto (SourceForge, 2008) se pudo constatar que presenta como principales características la posibilidad de usar tres distintos paradigmas de programación: basado en reglas, orientado a objetos y procedural; la portabilidad, debido al lenguaje en que está escrito puede usarse en diferentes sistemas operativos; y la integración/extensibilidad, ya que puede estar embebido dentro de código procedural y la verificación/validación.

- Drools

Es un sistema de gestión de reglas de negocio (BRMS, del inglés Business Rules Management System) de código abierto, desarrollado por JBoss con un motor de reglas que en sus versiones iniciales estaba basado en inferencia de encadenamiento hacia adelante. A partir del Sistema de Reglas de Producción (PRS, del inglés Production Rules System), Drools 5.X introdujo el estilo de razonamiento de Prolog de encadenamiento hacia atrás. Además implementa y extiende el algoritmo Rete, lo que significa que tiene una implementación mejorada y optimizada de dicho algoritmo para los sistemas orientados a objetos (The JBoss Team, 2013).

La adaptación de Rete a una interfaz orientada a objetos permite una mayor expresión natural de reglas de negocio en lo que respecta a los objetos de negocio. Además, establece la lógica de programación declarativa y es tan flexible como para permitir que coincida con la semántica del dominio del problema. Además soporta el estándar JSR-94 para su motor de reglas de negocio y usa JCR (Jackrabbit) para gestionar el repositorio de reglas, y el estándar JAAS (del inglés Java Authentication and Authorization Service) para la autorización y autenticación. Es un motor de inferencia bajo la licencia ASL (Apache Software License), gratuita y de libre utilización.

1.6.3. Selección del motor de inferencia

Para la selección del motor de inferencia a utilizar se tuvo en cuenta que la herramienta que lo proporcione sea preferiblemente software libre y que realice el proceso de inferencia utilizando la estrategia de

encadenamiento hacia adelante y hacia atrás, debido a que en un diagnóstico médico el especialista puede desear realizar un diagnóstico a partir de las características que presente el paciente (encadenamiento hacia adelante) o comprobar si a partir de una enfermedad propuesta el paciente puede padecerla (encadenamiento hacia atrás). También debe contar con soporte por parte de sus desarrolladores y consecuentemente que exista documentación amplia y actualizada de la misma. En la Tabla 5 se hace una comparación de las herramientas mencionadas.

Tabla 5. Comparación entre herramientas para la selección del motor de inferencia.

Nombre	**Licencia**	**Dirección de búsqueda**	**Soporte**	**Documentación actualizada**	**Otras características**
JESS	1. Sin costo para uso académico. 2. Licencia comercial por la que se debe pagar.	Hacia atrás y hacia adelante	Si	Si	
CLIPS	Software de dominio público.	Hacia atrás	No	No	
JEOPS	LGPLv2	Hacia adelante y hacia atrás	Si	Si	
Drools	ASL	Hacia adelante y hacia atrás	Si	Si	1. Integración mediante un *plugin* con el IDE Eclipse. 2. Posibilidad de gestionar las reglas de producción a través de una interfaz gráfica que brinda el "*Guvnor*", sistema gestor del repositorio de reglas.

A partir de la comparación realizada se determina usar el motor de inferencia brindado por el Sistema Gestor de Reglas del Negocio Drools en su versión 5.5.0 Final por cumplir con los requisitos planteados anteriormente.

1.6.4. Lenguaje de programación: Java 1.6

Java es un lenguaje de programación de propósito general, concurrente, basado en clases y orientado a objetos. Las aplicaciones desarrolladas usando Java son compiladas a código bytes y este es interpretado por la Máquina Virtual de Java, lo cual lo define como un lenguaje interpretado (Gosling, y otros, 2013). Se escoge debido a que Drools trabaja sobre él, incluso en la implementación de las bases de conocimiento para sistemas que usan Drools, existe una mezcla entre el lenguaje definido por este y el lenguaje Java. Otro aspecto importante es que los desarrolladores poseen experiencias anteriores en el desarrollo de aplicaciones usando Java.

1.6.5. Plataforma de desarrollo: Eclipse Indigo Service Release 2

Eclipse es una herramienta creada por la comunidad Eclipse, de desarrollo de software libre y según (The Eclipse Foundation, 2013) esta versión específica fue liberada el 22 de junio de 2011. Entre sus características principales cuenta con la plataforma principal, donde se ejecutan los módulos que posee; el SWT (del inglés Standard Widget Toolkit); el JFace, que provee el manejo de archivos y textos y editores de textos y el Workbench, que provee vistas, editores, perspectivas y asistentes. En él se pueden crear aplicaciones usando lenguajes como Java, C/C++ y Python. Además provee marcos de trabajos enriquecidos para el desarrollo de aplicaciones gráficas, por ejemplo, el marco de trabajo para la edición de gráficos (GEF, del inglés Graphics Editing Framework) diseñado para el desarrollo de editores visuales. Liberado bajo la licencia Eclipse Public License (Licencia Pública de Eclipse), los receptores de esta herramienta pueden utilizar, modificar, copiar y distribuir el trabajo y las versiones modificadas.

1.6.6. Sistema de gestión de base de datos: SQLite 3.7.2

SQLite implementa una base de datos SQL embebida, transaccional y sin necesidad de ser instalada o configurada para poder usarse. Su código está bajo dominio público, de esta forma puede ser usado para cualquier propósito: comercial o privado (SQLite, 2013). Puede ser usado en modo ventana de comando (Shell) o embebido en aplicaciones de código como C, C++ y Java y cumple con la mayoría de los estándares SQL92. Cuando se usa de forma embebida en aplicaciones, posee una gran ventaja: la portabilidad, sin necesidad de establecer un servidor de base de datos para la gestión de la información.

Esta es la principal característica por la que se optó por SQLite, evitando así la necesidad de instalar un servidor de bases de datos en el entorno donde se despliegue el software, ya que es una aplicación de escritorio y no necesita intercambiar información con otros clientes mediante una base de datos única.

1.7. Patrones arquitectónicos y de diseño.

En el desarrollo de sistemas informáticos es muy común el uso de patrones, debido a que expresan la solución a problemas que se repiten en distintas situaciones. Un patrón es una solución probada a un determinado tipo de problema y de acuerdo con (Sommerville, 2007) está compuesto por cuatro elementos: nombre, descripción del problema que resuelve, la solución del problema y resultados o consecuencias de la aplicación del patrón.

1.7.1. Patrones arquitectónicos

Según (Buschmann, y otros, 1996), un patrón arquitectónico expresa esquemas para la organización estructural fundamental para sistemas de software. Ellos proveen un conjunto de subsistemas predefinidos, especifican sus responsabilidades, e incluye reglas y guías para organizar la relación entre ellos. Algunos autores como (Garlan, y otros, 1994), plantean conceptos muy similares haciendo alusión a estilos arquitectónicos. En el presente trabajo se asume el concepto planteado por Buschmann como patrón arquitectónico.

Patrón N-Capas

Cuando se define una arquitectura en capas para un sistema, este se estructura en grupos de subtareas (capas) en las cuales cada uno representa un nivel particular de abstracción. Este patrón generalmente es utilizado en el desarrollo de aplicaciones de escritorio, donde cada capa define un conjunto de funciones o servicios, que son consumidos o utilizados por la capa inmediata superior, y esta, a su vez, utiliza los servicios que le proporciona la capa inmediata inferior, como se muestra en la Figura 5.

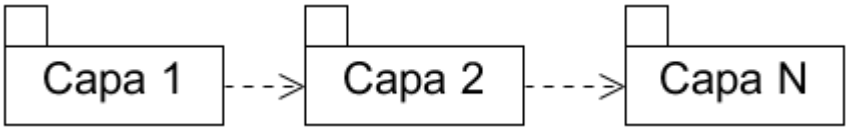


Figura 5. Estructura general de un sistema con arquitectura en capas.

En la figura anterior la Capa 1 consume los servicios que brinda la Capa 2 y para acceder a algún servicio de la Capa N debe hacerlo a través de la Capa 2 (capa intermedia). Este patrón conviene ser usado en

grandes aplicaciones que requieren de descomposición y se debe tener en cuenta los siguientes elementos (Buschmann, y otros, 1996):

- Definir los criterios de abstracción.
- Determinar el número de niveles de abstracción (capas) de acuerdo al criterio de abstracción.
- Nombrar las capas y asignarles tareas a cada una.
- Especificar los servicios.
- Refinar las capas.
- Especificar una interfaz para cada una de las capas.
- Estructurar las capas individualmente cuando esta es muy compleja.
- Especificar la comunicación entre capas adyacentes.
- Diseñar una estrategia de manejo de errores.

El patrón N-Capas tiene varios beneficios, entre ellos el reúso de las capas, el soporte a la estandarización, la dependencia se mantiene de forma local y la intercambiabilidad ya que implementaciones de capas individuales pueden ser reemplazadas por implementaciones semánticamente equivalentes sin realizar grandes esfuerzos.

1.7.2. Patrones de diseño

Patrones GRASP

En los patrones GRASP (del inglés General Responsibility Assignment Software Patterns), se plantean los principios generales para la asignación de responsabilidades a objetos (Larman, 1999). Ellos son:

- Experto: Ayuda a determinar cuándo determinado objeto debe cumplir cierta responsabilidad (ejecutar una acción, activar un estado, etc.). Para la ejecución del sistema en su totalidad todas las clases deben colaborar a través de mensajes para realizar determinadas acciones, cuál debe realizar cada acción está determinado por la información que posee cada una. Este patrón ayuda a mantener el encapsulamiento, dando soporte a un bajo acoplamiento, lo que favorece la robustez del sistema y el fácil mantenimiento. Además con la aplicación de estos principios se obtienen definiciones de clases más sencillas y más cohesivas, dando soporte a una alta cohesión.
- Creador: Ayuda a establecer quién es el responsable de la creación de una instancia de determinada clase. Plantea que una clase *A* debe ser responsable de instanciar una clase *B* si: *A* agrega, contiene

o utiliza específicamente los objetos de *B*; *A* registra las instancias de los objetos de *B* o *A* tiene los datos de inicialización que serán transmitidos a *B* cuando este objeto sea creado.

- Bajo Acoplamiento: Este patrón ayuda a disminuir el acoplamiento (nivel en que una clase está conectada a otras), proporcionando una dependencia escasa entre clases y un aumento de la reutilización. De esta manera se garantiza que no deban realizarse cambios locales en ellas por cambios producidos en las clases afines, que sean menos difíciles de entender cuando estén aisladas y que puedan ser más reutilizables.
- Alta Cohesión: Desde la perspectiva del diseño orientado a objetos, la cohesión funcional es una medida de cuan relacionadas están las responsabilidades de una clase. Cuando estas realizan funcionalidades estrechamente relacionadas existe una alta cohesión, proporcionando una mejor reutilización y comprensión de la clase en sí.
- Controlador: Este patrón define cuándo una clase debe atender un evento del sistema. Teniendo en cuenta la alta cohesión y el bajo acoplamiento se deben diseñar clases controladoras que realicen las operaciones para dar respuesta a las acciones que ejecuten los usuarios y a su vez, puedan ser reutilizadas. Con este principio se logra separar la lógica del negocio (dominio) de otras clases (por ejemplo, interfaces) que rara vez son reutilizables por sus características propias.
- Fabricación Pura: Este patrón plantea el principio de asignación de un conjunto de responsabilidades altamente cohesivo a una clase artificial, que no representa nada en el dominio del problema, pero es necesaria su creación en aras de mantener un bajo acoplamiento y una alta cohesión.
- No Hables con Extraños: Este patrón se usa para no acoplar una clase *A* que consume servicios de otra *B*, que es indirecta. Entiéndase por indirecta que no es un objeto propio de la clase *A*, no es un parámetro de una operación en *A,* no es un atributo propio de *A*, no es un elemento de una colección que sea atributo propio de *A* y no es un objeto creado en el interior de la operación en *A*. Con este principio se garantiza no acoplar la clase *A* al conocimiento del objeto indirecto *B*, pues si un objeto conoce las conexiones internas y las estructuras de otros entonces presenta un acoplamiento alto.
- Polimorfismo: Este patrón ayuda a manejar las alternativas o comportamientos afines basados en el tipo y facilita la agregación de extensiones futuras que requieren las variaciones imprevistas.
- Indirección: Ayuda a definir qué elemento debe cumplir cierta responsabilidad a fin de evitar el acoplamiento directo, organizando los elementos de forma que se logre un bajo acoplamiento. Plantea que se debe asignar la responsabilidad a un objeto intermedio para que medie entre otros componentes o servicios creando de esta forma una “indirección”.

Patrones GoF

Los patrones GoF (del inglés Gang of Four) que significa Pandilla de los Cuatro, son patrones de diseño que describen soluciones a problemas específicos en el diseño de software orientado a objetos. En total son 23 patrones descritos en (Gamma, y otros, 1994) y se clasifican en patrones creacionales, estructurales y de comportamiento. En el primer grupo están los patrones que describen los principios generales relacionados con el proceso de creación de objetos y clases, encargándose de abstraer el proceso de instanciación. En este grupo se encuentran los patrones Fábrica Abstracta, Constructor, Método Fábrica, Prototipo, Instancia Única. En el segundo grupo están los patrones concernientes a cómo las clases y los objetos están compuestos para formar estructuras más complejas. Ellos son: Adaptador, Puente, Composición, Decorador, Fachada, Peso Mosca y Apoderado. Por último, en el grupo de los patrones de comportamiento, se encuentran los concernientes a los algoritmos y la asignación de responsabilidades entre objetos. Estos patrones son Cadena de Responsabilidades, Acción, Intérprete, Iterador, Mediador, Recuerdo, Observador, Estado, Estrategia, Método Plantilla y Visitante.

Conclusiones del capítulo

A partir de las características y necesidades que posee el CNGM, se determinó la creación de un SE para el diagnóstico de enfermedades genéticas. Para la creación del sistema SEDEGEN se definió como metodología de desarrollo la propuesta por Weiss y Kulikowski debido a que sus características pueden ser aplicadas al contexto del desarrollo del sistema. Además se seleccionó el lenguaje de modelado UML, ya que mediante él se pueden crear diagramas que reflejan las estructuras estáticas y dinámicas para el modelado de cualquier sistema orientado a objetos. Se decidió emplear el Sistema de Gestión de Reglas del Negocio Drools principalmente por las potencialidades de su motor de inferencia, proporcionando ambas direcciones de búsqueda (hacia atrás y hacia delante) en el proceso de inferencia. En función de esa herramienta se seleccionó el lenguaje de programación Java y el entorno de trabajo Eclipse para realizar la implementación del SE. Por último, se escogió SQLite como sistema de gestión de base de datos debido a la posibilidad de incluir una base de datos embebida en el proyecto, sin necesidad de instalar un sistema gestor de base de datos independiente. Mediante el uso de patrones arquitectónicos y de diseño se logra desarrollar un sistema aplicando buenas prácticas de desarrollo de software.

Capítulo 2: Análisis y diseño del sistema experto

Este capítulo está estructurado de acuerdo a la metodología seleccionada. En cada acápite se abordan los elementos correspondientes a las tres primeras fases de la misma: Planteamiento del Problema, Obtención del Conocimiento Experto y Diseño del Sistema Experto. Además, para guiar el proceso de análisis y diseño, se toman elementos de los principios generales del desarrollo de software, planteados en libros como (Pressman, 2010) y (Sommerville, 2007), aplicables a la mayoría de los proyectos de desarrollo de software. De esta forma se hace la obtención de requisitos y se generan artefactos como el modelo conceptual, diagrama de casos de uso, la vista lógica del sistema para representar la arquitectura del mismo, y modelo del diseño, haciendo uso de los patrones arquitectónico y de diseño.

2.1. Planteamiento del problema

Para realizar un análisis de la situación del CNGM respecto al diagnóstico de las enfermedades genéticas se efectuaron entrevistas con el cliente. Se determinó que los especialistas, principalmente los de menor experiencia laboral, la mayoría de las veces necesitan consultar información procedente de libros especializados que le aporten información útil para poder realizar las diagnosis. La forma de hacerlo es la tradicional, buscando la información página a página, sin ninguna forma automatizada, lo que retrasa el dictamen final. Otra forma de obtener información es buscando apoyo en diferentes especialistas de más experiencia, pero igualmente esto exige tiempo que puede ser empleado para determinar elementos particulares del diagnóstico relacionados con el paciente y en los que realmente debe hacerse énfasis. La tarea de consulta de información es un elemento trivial, comparado con la valoración que debe hacer el especialista de ella. Por tal razón en el centro se plantea la necesidad de automatizar la búsqueda de información de enfermedades genéticas y además tener disponible para cada especialista un diagnóstico de apoyo sin la necesidad de consultar otros colegas.

Para tener un mejor entendimiento del problema se describe en un modelo conceptual (Figura 6) los principales elementos que intervienen en el proceso de diagnóstico de un especialista a un paciente. Un modelo conceptual *"(…) es una representación de conceptos en un dominio del problema*" (Larman, 1999). En este tipo de modelo se ilustra, haciendo uso del lenguaje UML, un conjunto de diagramas de estructura estática que permite mostrar conceptos significativos en el dominio del problema, asociaciones entre estos conceptos y sus atributos, aunque estos últimos no necesariamente deben representarse.

Capítulo 2

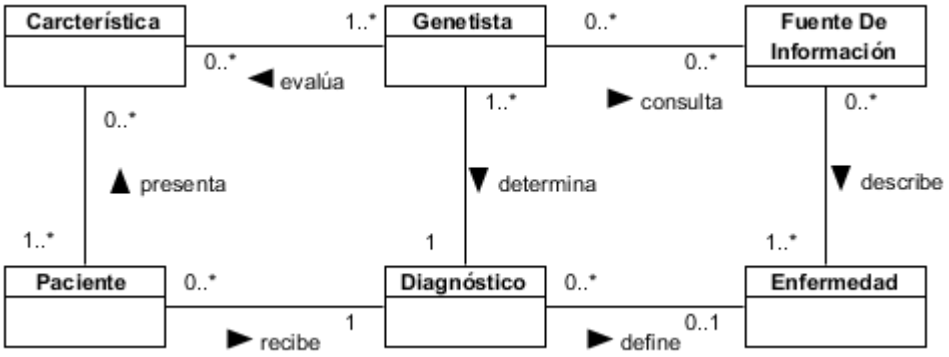


Figura 6. Modelo conceptual.

A continuación se explican cada uno de los elementos representados:

- Paciente: Es la persona que va al doctor en busca de ayuda especializada y recibe un diagnóstico médico. En dependencia de si presenta o no síntomas que indiquen que tiene determinada enfermedad genética se le diagnostica o no tal enfermedad.
- Característica: Representa los signos, síntomas y estructuras afectadas que se conocen actualmente que son provocados por enfermedades genéticas.
- Genetista: Es el especialista que atiende a un paciente. Este debe identificar qué características de las que posee el paciente pueden ser anormales y a su vez ser causadas por alguna enfermedad genética. En dependencia de estos factores brinda un diagnóstico.
- Diagnóstico: Es el veredicto del genetista y plantea si el paciente posee o no alguna de las enfermedades genéticas conocidas.
- Fuente de información: Debido al gran número de enfermedades genéticas existentes y a la rareza de la mayoría de ellas, muchas veces el genetista debe apoyarse en fuentes de información que le ayuden a realizar un diagnóstico. Estas fuentes de información pueden ser libros, sitios online u otros especialistas.
- Enfermedad: Es el padecimiento que causa que el paciente presente determinadas características fuera de lo común. Para asociar una enfermedad a un paciente se deben analizar las características del mismo y emitir un diagnóstico corroborando el padecimiento o no de la enfermedad.

2.1.1. Requisitos del sistema

Plantea (Sommerville, 2005) que los requisitos "*(…) son la descripción de los servicios proporcionados por el sistema y sus restricciones operativas*", estos, además, reflejan las necesidades de los clientes de un

sistema que ayude a resolver algún problema. Los requisitos pueden ser funcionales o no funcionales. Los requisitos funcionales *"(…) son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que este debe reaccionar a entradas particulares y de cómo debe comportarse en determinadas situaciones particulares (…)"* (Sommerville, 2005). Por otra parte los requisitos no funcionales "*son restricciones de los servicios o funciones ofrecidos por el sistema (…)"* (Sommerville, 2005) y pueden ser restricciones sobre el proceso de desarrollo, restricciones de tiempo y estándares.

A partir de la identificación del problema planteado anteriormente, se definieron una serie de características con las que debe cumplir el sistema para darle solución. De esta forma se delimitaron los siguientes requisitos funcionales y no funcionales.

Requisitos funcionales

RF 1. Realizar diagnóstico de enfermedades por signos, síntomas y/o estructuras afectadas.

RF 2. Realizar búsqueda de enfermedades por referencias bibliográficas.

RF 3. Realizar búsqueda de enfermedades por bases genéticas.

RF 4. Realizar búsqueda de enfermedades por nombre.

RF 5. Visualizar datos de enfermedades.

RF 6. Configurar apariencia.

RF 7. Configurar búsqueda por nombre.

RF 8. Configurar diagnóstico.

Requisitos no funcionales

- Apariencia o interfaz externa

  RNF 1. El sistema debe poseer los colores establecidos por el Ministerio de Salud Pública (color verde, y otros colores claros que contrasten con el verde).

- Usabilidad

  RNF 2. En la interfaz principal debe haber un menú principal con todas las opciones que brinda el sistema.

RNF 3. La interfaz principal también debe tener una barra de herramientas con las acciones más comunes en el sistema para agilizar el acceso a las mismas.

RNF 4. Los botones de las ventanas secundarias deben estar ubicados en la esquina inferior derecha, en el caso de los botones *Aceptar,* deben estar ubicados a la izquierda y los *Cancelar* a la derecha, en caso de existir botones *Aplicar*, estos deben estar ubicados entre los *Aceptar* y *Cancelar*. Los botones de *Minimizar*, *Maximizar* y *Cerrar* deben estar ubicados en la esquina superior derecha de las interfaces.

RNF 5. Las interfaces deben tener un botón *Ayuda* en la parte superior que explica las operaciones que se realizan en esa interfaz.

RNF 6. Los botones deben poseer descripciones (tooltips) que muestren información de las acciones que realizan.

RNF 7. La aplicación puede ser usada por especialistas en genética médica con conocimientos básicos en el manejo de las computadoras.

- Rendimiento

RNF 8. Los tiempos de respuesta deben ser los siguientes: para las búsquedas no mayores de 4 segundos y para los diagnósticos no mayores de 24 segundos.

- Software

RNF 9. La computadora donde se despliegue la aplicación debe tener instalado el sistema operativo Windows (7, 8 u 8.1) y/o Linux y la Máquina Virtual de Java (1.6 como mínimo).

- Hardware.

RNF 10. La computadora donde se despliegue la aplicación debe poseer 1 GB de memoria RAM como mínimo, microprocesador Intel(R) Pentium(R) 4 o superior, y 1GB en disco duro disponible.

- Requisitos de Licencia

RNF 11. Las herramientas y las tecnologías en que estará basada la aplicación informática deben cumplir con las licencias de software libre.

- Ayuda y documentación

RNF 12. El sistema deberá brindar una ayuda para el mejor entendimiento del sistema, así como para utilizar correctamente sus funcionalidades, donde se explique cada una de las acciones que se puedan realizar y cómo deben hacerse.

2.1.2. Diagrama de casos de uso

Mediante los casos de uso (CU) se representan un conjunto de escenarios que describen cómo el software va a ser usado en una determinada situación. Con esta intención se definieron a partir de los requisitos funcionales los siguientes casos de uso, como se muestra en la Figura 7, proporcionando una vista común a desarrolladores y clientes de las funcionalidades que proveerá el sistema.

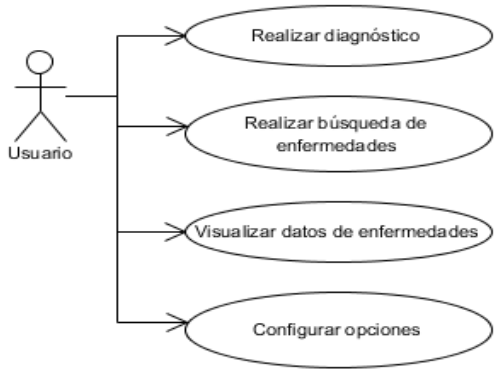


Figura 7. Diagrama de casos de uso del sistema.

En el diagrama anterior el *Usuario* inicia los casos de uso *Realizar diagnóstico*, *Realizar búsqueda de enfermedades, Visualizar datos de enfermedades y Configurar opciones*. El primero representa la acción del *Usuario* al solicitar al sistema ejecutar un diagnóstico por determinado criterio. El segundo se inicia cuando el actor solicita realizar una búsqueda de enfermedades por alguno de los criterios siguientes: nombre, referencias bibliográficas o bases genéticas, sin necesidad de realizar el diagnóstico de un paciente. El tercer CU se inicia cuando el *Usuario* desea visualizar toda la información referente a alguna(s) enfermedad(es) después de realizado un diagnóstico o una búsqueda. El último CU representa la acción del *Usuario* cuando este solicita al sistema configurar las opciones por las cuales se realizan las acciones de búsqueda de enfermedades y diagnósticos. En la Tabla 6 se realiza la especificación formal del CU arquitectónicamente significativo *Realizar diagnóstico*.

Para consultar la descripción de los restantes casos de uso debe remitirse al expediente de proyecto, específicamente a la plantilla *010114a_Especificacion_de_casos_de_uso_SEDEGEN.doc*.

Tabla 6. Especificación formal del CU *Realizar diagnóstico*

Caso de Uso	Realizar diagnóstico
Objetivo	Realizar un diagnóstico de posibles enfermedades a partir de signos, síntomas y/o características que presente un paciente.
Actores	Usuario
Resumen	El caso de uso se inicia cuando el usuario selecciona la opción “Realizar diagnóstico”, la cual consiste en determinar las enfermedades que más probabilidades tienen de ser padecidas por el paciente, partiendo de los signos y síntomas y/o estructuras del cuerpo afectadas que presenta el mismo (seleccionadas por el *Usuario* en la interfaz del sistema) y finaliza cuando el sistema muestra un listado con el nombre de las enfermedades que puede padecer el paciente.
Referencias	RF 1
Prioridad	Crítico
Flujo Normal de Eventos	
Flujo básico “Realizar diagnóstico”	
Acción del actor	**Respuesta del sistema**
1. El caso de uso se inicia cuando el usuario selecciona la opción “Realizar diagnóstico”.	2. El sistema muestra la interfaz con los signos y síntomas de todas las enfermedades que posee en su base de conocimiento, organizados por estructuras del cuerpo humano.
3. El usuario selecciona los signos y/o síntomas y/o las estructuras como criterio para realizar el diagnóstico (características que posee el paciente) y selecciona la opción “Aceptar”.	4. El sistema verifica que se haya seleccionado al menos un elemento.

	5. El sistema muestra un mensaje de confirmación para realizar el diagnóstico.
6. El usuario selecciona la opción “Si”.	7. El sistema muestra una interfaz con el resultado de la inferencia: la enfermedad más probable y la explicación de los elementos que se tuvieron en cuenta para seleccionarla; y un listado de los nombres de otras enfermedades a ser posibles diagnósticos. Terminando así el caso de uso.

Prototipo de Interfaz

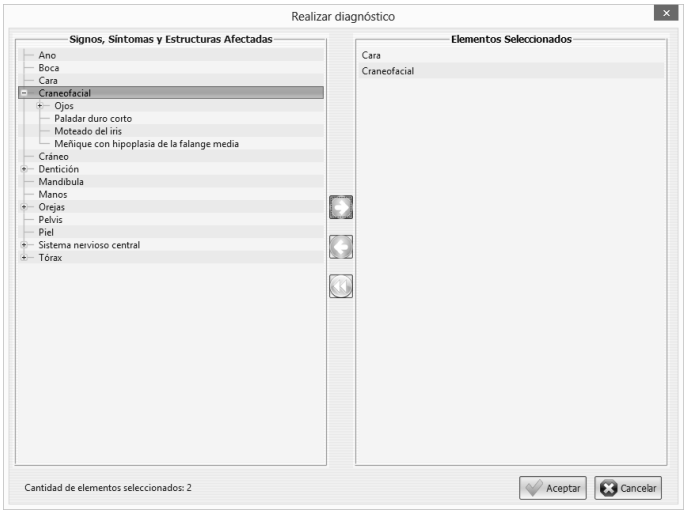


Figura 8. Prototipo de interfaz para realizar diagnósticos.

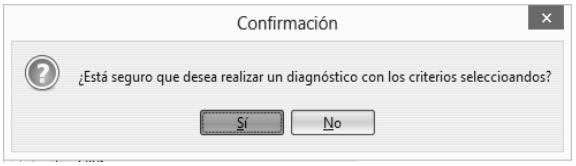


Figura 9. Prototipo de interfaz de confirmación para realizar diagnósticos.

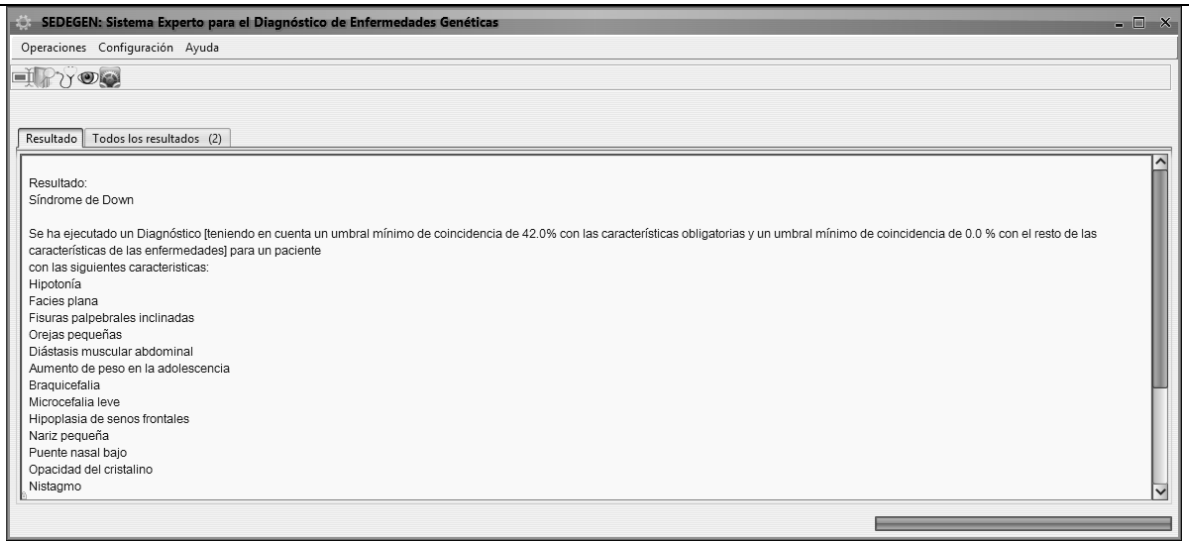


Figura 10. Prototipo de interfaz para mostrar el resultado del diagnóstico.

Flujos Alternos	
Flujo alterno al paso 3 "Diagnóstico cancelado"	
3.a El usuario selecciona la opción "Cancelar".	3.b El sistema muestra una interfaz de confirmación.
3.c El usuario selecciona la opción "Si".	3.d El sistema cancela el diagnóstico y regresa a la interfaz principal. Terminando así el caso de uso.

Prototipo de Interfaz

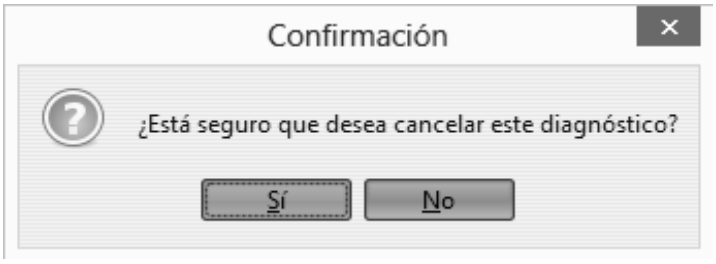


Figura 11. Prototipo de interfaz de confirmación para cancelar diagnóstico.

Flujo alterno al paso 5 " Selección vacía"	
	5.a El sistema muestra un mensaje informando que debe seleccionar al menos un elemento, regresando así al paso 2 del flujo básico "Realizar diagnóstico".

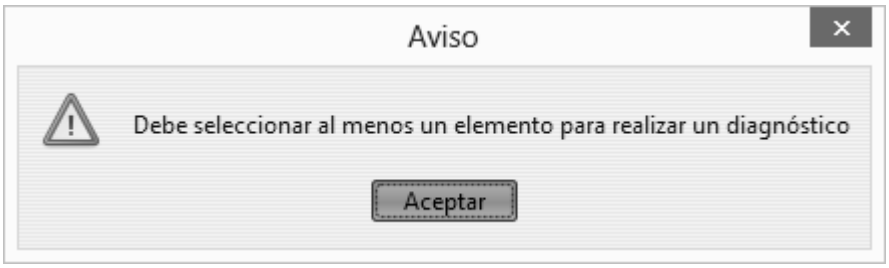
<table>
<tr><td colspan="2">Prototipo de interfaz
 

Figura 12. Prototipo de interfaz de aviso cuando no se ha seleccionado ningún elemento.</td></tr>
<tr><td colspan="2">Flujo alterno al paso 6 “Confirmación cancelada”</td></tr>
<tr><td>6.a El actor selecciona la opción “No”.</td><td>6.b El sistema regresa al paso 2 del flujo básico “Realizar diagnóstico”.</td></tr>
<tr><td colspan="2">Prototipo de interfaz
Ver prototipo de interfaz del flujo alterno 3.</td></tr>
</table>

2.2. Obtención del conocimiento experto

Como elemento indispensable para la creación del SBR que aborda el presente trabajo, se debe realizar la elaboración de una BC que almacene todas las reglas de producción a ser usadas por el MI para realizar la inferencia que brinde un posible diagnóstico. Para la creación de la BC que usará el sistema, los genetistas del CNGM juegan un papel importante, pues son los que poseen la experiencia y la información relacionada con el diagnóstico de estas enfermedades. Además es importante contar con literatura especializada en el tema.

Para llevar a cabo el proceso de obtención del conocimiento se contó, específicamente, con la información brindada por el DrC. Roberto Lardoeyt Ferrer. Para la obtención del conocimiento se efectuaron reuniones informales y entrevistas entre el experto y el equipo de desarrollo, donde quedaron especificadas las enfermedades que el sistema debe ser capaz de identificar, los signos y síntomas que presentan los pacientes que padecen tales enfermedades y las estructuras del cuerpo que son dañadas por ellas. También se identificaron particularmente algunos de estos síntomas que no se presentan en todos los casos de las enfermedades que los provocan, analizando además la frecuencia con que se exhiben. Además se utilizó el libro “*SMITH. Patrones reconocibles de malformaciones humanas*” (Jones, 2007) como fuente

bibliográfica con la asesoría del experto. El libro contiene la descripción detallada de las enfermedades genéticas más frecuentes, en total una descripción de 285 enfermedades genéticas.

2.3. Diseño del Sistema Experto

En la literatura especializada el término arquitectura de software puede ser encontrado con varias definiciones, pues no se ha establecido un estándar que pueda ser tomado como referencia. A pesar de esto la idea esencial en cada planteamiento es que la arquitectura de software constituye la organización fundamental de un sistema, los componentes que conforman el mismo, así como la relación entre ellos. El presente documento se acoge al concepto de arquitectura brindado por la documentación de Microsoft como sigue:

"*La arquitectura de software abarca el conjunto de decisiones significantes acerca de la organización de un sistema de software incluyendo la selección de elementos estructurales y sus interfaces por la cual el sistema está compuesto; el comportamiento especificado en la colaboración entre esos elementos; la composición de estos en cuanto a estructura y comportamiento dentro de grandes subsistemas; y un estilo arquitectónico que guía esta organización. La arquitectura de software también involucra funcionalidad, usabilidad, resistencia, rendimiento, reúso, comprensibilidad, restricciones tecnológicas y económicas, equilibrio y preocupaciones estéticas*" (Microsoft, 2014).

2.3.1. Patrón arquitectónico utilizado

En la construcción del sistema se aplicó el patrón arquitectónico N-Capas, donde se definieron 3 niveles de abstracción (capas): Modelo, Lógica y Presentación. A continuación se explica cada una en la Figura 13, utilizando como apoyo la vista lógica del modelo 4+1 vistas propuesto por Kruchten en (Kruchten, 1995), la cual es usada para representar el modelo de objetos del diseño.

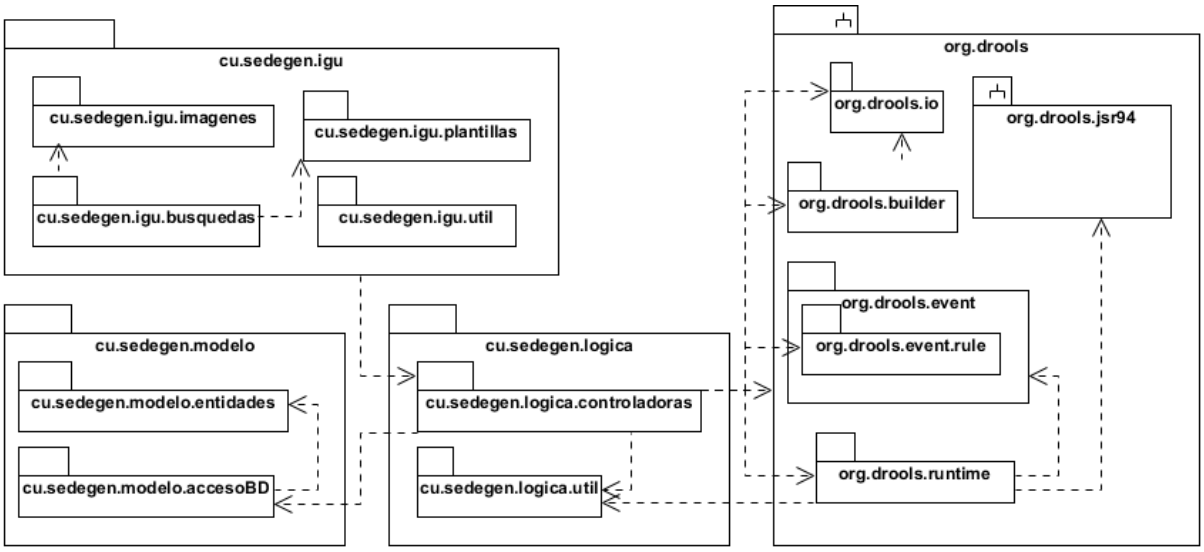


Figura 13. Vista lógica general del sistema.

En la figura anterior fueron representados los siguientes elementos:

Capa Modelo (*cu.sedegen.modelo*): Este nivel está compuesto por las clases que representan o manejan la información persistente del sistema, es decir, los datos de las enfermedades, que son mostrados al usuario una vez que se realiza alguna búsqueda, así como los procedimientos necesarios para acceder a dicha información. Esta capa brinda servicios a la superior (Capa Lógica) y está descompuesto en las siguientes subcapas (paquetes) aplicando el mismo principio de organización:

- *cu.sedegen.modelo.entidades*: Contiene las clases entidades, es decir las clases que son utilizadas para encapsular la información persistente de la base de datos.
- *cu.sedegen.modelo.accesoBD*: Contiene las clases necesarias para acceder a la base de datos. Es a través de una de estas clases que la capa superior consume los servicios de este nivel.

Capa Lógica (*cu.sedegen.logica*): Esta capa contiene las estructuras, clases y procedimientos necesarios para dar cumplimiento a los requisitos y también está constituida por subcapas para lograr una mayor organización. Utiliza las funcionalidades brindadas por el subsistema Drools para realizar el proceso de inferencia y de explicación. Además usa los servicios que brinda la capa Modelo para obtener datos de las enfermedades cuando se ejecutan procedimientos de búsquedas. Brinda funcionalidades a la capa Presentación y está estructurada en los siguientes paquetes:

- *cu.sedegen.logica.controladoras*: Contiene las clases controladoras, son las clases que en colaboración implementan las funcionalidades que dan respuesta a los requisitos del software.

- *cu.sedegen.logica.util*: Contiene clases auxiliares que brindan funcionalidades que no definen la respuesta a los requisitos, pero son necesarias y utilizadas por los procedimientos principales de las clases controladoras. Esta organización permite la reutilización de código y una mejor estructura de la capa en sentido general.

Capa Presentación (*cu.sedegen.igu*): En este nivel están organizadas las interfaces gráficas de usuario también en subcapas y de una forma bien estructurada, de acuerdo a su función. Estas acceden a los procedimientos definidos por la capa inferior (Lógica) para obtener los servicios que la misma brinda y mostrarle al usuario las respuestas requeridas una vez realizadas las peticiones al sistema. En este paquete se encuentra la interfaz principal, a través de la cual se accede a los servicios de la capa Lógica y se estructura de la siguiente forma:

- *cu.sedegen.igu.busquedas*: Contiene interfaces gráficas que le muestran al usuario opciones de búsquedas de enfermedades por diferentes criterios.
- *cu.sedegen.igu.plantillas*: Contiene interfaces reutilizables por las interfaces principales de la aplicación.
- *cu.sedegen.igu.imagenes*: Contiene imágenes utilizadas por los componentes visuales de la aplicación.
- *cu.sedegen.igu.util*: Contiene interfaces auxiliares que colaboran con el funcionamiento del sistema en la capa Presentación.

Subsistema Drools (*org.drools*): Es el subsistema usado para la gestión de reglas y no comprende en si una capa en la estructura de la aplicación, sino que es un subsistema independiente; pero es necesario hacer notar cada uno de los componentes que posee enmarcados en la arquitectura de un SE. En este paquete de forma general se lleva cabo el control de coherencia mediante el Sistema de Mantenimiento de Verdad de Drools.

- *org.drools.builder*: Contiene las clases necesarias para cargar los ficheros correspondientes a la BC que usa el sistema para realizar el proceso de inferencia.
- *org.drools.event.rule*: Contiene clases usadas para construir el subsistema de explicación. Además posee clases que brindan la posibilidad de elaborar el subsistema de adquisición de conocimiento.
- *org.drools.runtime*: Contiene clases usadas en la construcción del subsistema de ejecución-acción.
- *org.drools.jsr94*: Este subsistema es una implementación del motor de inferencia usando java.

2.3.2. Diagrama de clases del diseño

Partiendo de los casos de uso del sistema se realizaron los correspondientes diagramas de clases del diseño (DCD). Mediante este tipo de diagramas se reflejan qué elementos del software interactúan para la realización de tales casos de uso, mostrándose las clases con sus operaciones y la colaboración entre ellas para dar respuesta a los requisitos. En la Figura 14 se muestra el DCD para el CU *Realizar diagnóstico*.

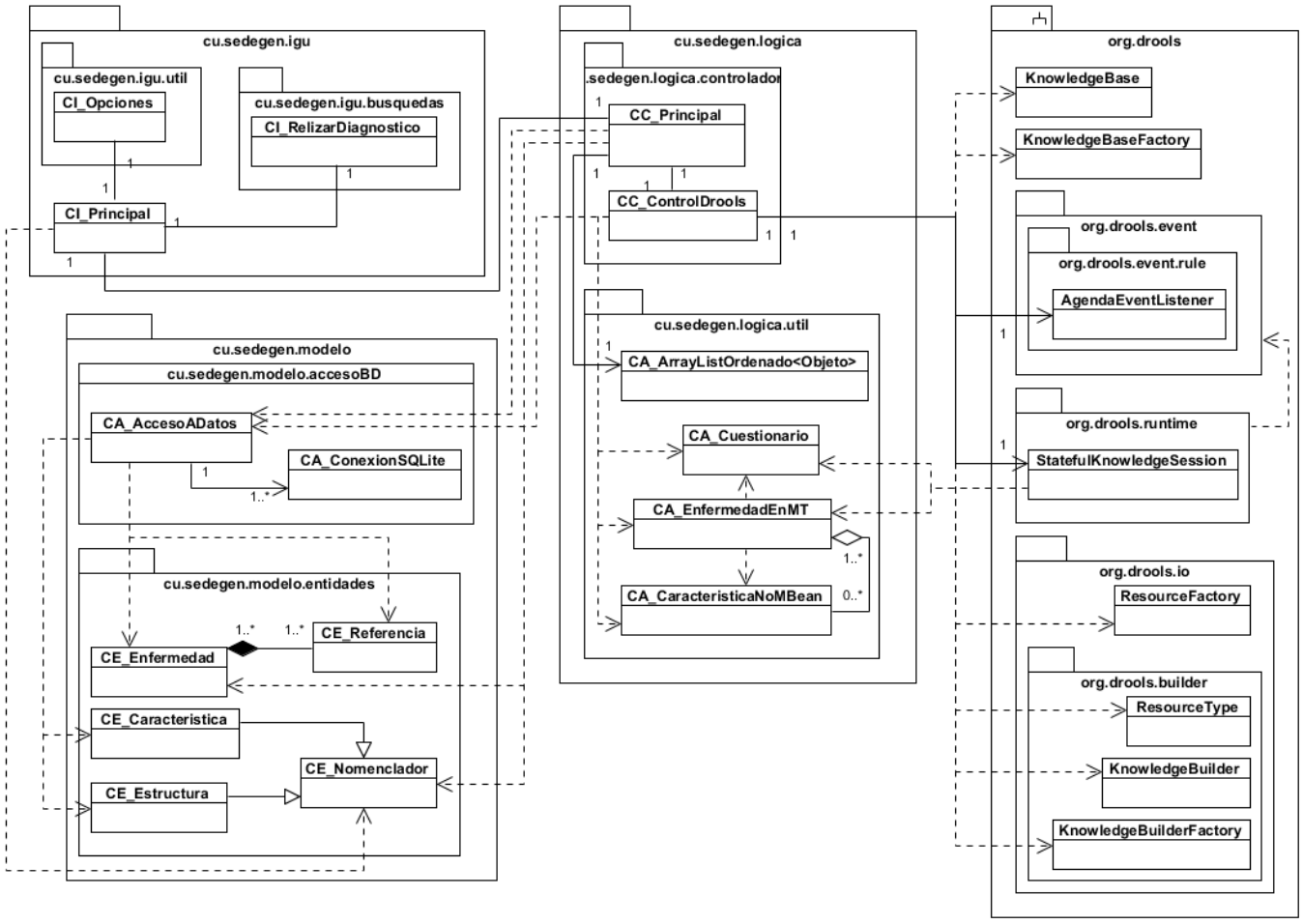


Figura 14. Diagrama de clases del diseño para el CU *Realizar diagnóstico*.

La clase interfaz *CI_RealizarDiagnostico* es la encargada de enviarle los valores de inferencia (signos, síntomas y/o estructuras afectadas) a la clase interfaz *CI_Principal*, seleccionados por el usuario. La clase *CI_Principal* obtiene de la clase *CI_Opciones* dos umbrales: el primero es la probabilidad mínima que se debe tener en cuenta para mostrar una enfermedad como posible diagnóstico para un paciente teniendo en cuenta las *características obligatorias*, el segundo umbral es análogo al primero pero teniendo en cuenta

las *características ocasionales* (tanto los umbrales como otras configuraciones, son cargados automáticamente de un fichero por la clase *CI_Opciones* al iniciar el sistema). Luego de adquiridos los umbrales, la clase *CI_Principal* invoca el procedimiento de la clase controladora *CC_Principal* para realizar un diagnóstico con los parámetros obtenidos. Esta a su vez, invoca el procedimiento de la clase controladora *CC_ControlDrools* para realizar la inferencia usando el subsistema Drools (*org.drools*).

Una vez que se hayan analizado todas las reglas y el proceso de inferencia haya terminado, se buscan los datos de las enfermedades en la base de datos *SQLite* a través de la clase *CA_AccesoADatos* y se devuelven en forma de objetos mediante las clases *CE_Enfermedad y CE_Referencia*. Después de obtenida la información, en *CC_Principal* se ordena el listado de enfermedades de acuerdo a la probabilidad determinada en las reglas haciendo uso de la clase *CA_ArrayListOrdenado.*

En el anterior diagrama no se incluyeron las operaciones de las clases, para consultar una descripción más detallada del mismo, así como de los restantes diagramas de clases del diseño, debe remitirse al expediente de proyecto, específicamente a la plantilla *010215_Modelo_de_diseño_SEDEGEN.doc*.

2.3.3. Patrones aplicados

Patrones GRASP

Como buenas prácticas de desarrollo de software, para la construcción de los diagramas, se tuvieron en cuenta los patrones GRASP y GoF, garantizando un correcto diseño de la interacción de los objetos del sistema, así como la asignación de responsabilidades. A continuación se mencionan los patrones aplicados y su impacto en el sistema.

- Experto: Una de las aplicaciones de este patrón se refleja en la clase *CC_Principal,* la cual tiene asignada la responsabilidad de efectuar las operaciones relacionadas con las búsquedas de datos, porque ella posee la información de las búsquedas que se realizan, es decir debe brindar cantidad de resultados obtenidos, probabilidad de padecimiento del paciente de cada una de las enfermedades mostradas, así como los datos de estas.
- Creador: La instanciación de clases es la acción más común en la POO, por esta razón el patrón creador debe aplicarse al sistema, garantizando así un bajo acoplamiento. Una de sus aplicaciones se evidencia en la clase *CC_ControlDrools*, que es creadora de la clase *CA_Cuestionario.*
- Bajo Acoplamiento: En el sistema se logra un bajo acoplamiento aplicando los principios de este patrón y gracias a la aplicación de los demás patrones relacionados (Experto, Creador, etc.).

- Alta Cohesión: En el sistema se tuvo en cuenta los principios planteados por este patrón, pues las clases realizan solamente funciones relacionadas entre sí o enmarcadas en un área específica, por ejemplo la clase *CC_ControlDrools*, presenta funcionalidades relacionadas con el manejo del subsistema Drools, la clase *CA_AccesoADatos* posee funcionalidades relacionadas con las consultas a la base datos y así respectivamente, cada clase está destinada a la realización de tareas afines. Esto mejora la claridad y facilidad para entender el diseño, propicia que se simplifique el mantenimiento y la mejora de funcionalidades y genera además un bajo acoplamiento.
- Controlador: En el SE diseñado existen dos clases controladoras: *CC_Principal* y *CC_ControlDrools*, la primera se encarga de atender las peticiones del usuario a través de las interfaces gráficas y la segunda atiende las tareas que delega la controladora principal, relacionada con el manejo del subsistema Drools, para realizar otras operaciones.
- Fabricación Pura: En el sistema la clase *CA_AccesoADatos* no representa ningún elemento del dominio del problema, pero se hace necesaria su creación para que efectúe el manejo de las operaciones con la base de datos. De asignarle estas responsabilidades a alguna de las otras clases, se perdería la cohesión alta, el bajo acoplamiento, propiciando que los componentes sean menos reutilizables y más complejos de realizares mantenimiento y mejora de funcionalidades.
- No Hables con Extraños: La clase *CI_Principal*, que es una interfaz gráfica, para realizar una consulta a la base de datos, en lugar de invocar directamente un procedimiento de la clase *CA_AccesoADatos*, usa un objeto de la clase *CC_Principal* como intermediario, porque es un objeto directo de ella, ya que *CI_Principal* agrega a *CC_Principal*.
- Polimorfismo: En la Figura 15 se observa como el método abstracto *getTipo* se define en la clase *CE_Nomenclador* y debe ser redefinido en las clases hijas *CE_Estructura* y *CE_Caracteristica* de acuerdo al comportamiento específico en ellas.

Patrones GoF

- Método Fábrica: El objetivo de la aplicación de este patrón es definir la interfaz para la creación de un objeto, pero permitiendo a las subclases decidir qué clase instanciarlo. En la Figura 15 se observa como la clase abstracta *CE_Nomenclador* define los métodos *getIdBD* y *getNombre*, los cuales son invariantes para cualquier subclase de ella, además define el método abstracto *getTipo;* de esta forma la creación de un objeto *CE_Nomenclador* es exclusiva de las clases *CE_Caracteristica* y *CE_Estructura*, las cuales deben redefinir el método *getTipo*.

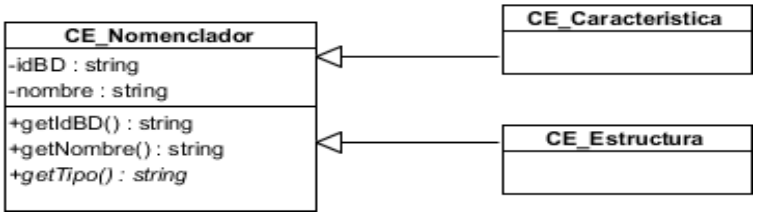


Figura 15. Ejemplo de aplicación del Método Fábrica.

- Instancia Única: El objetivo de la aplicación de este patrón es garantizar que una clase tiene una instancia única, brindando un punto de acceso global a la misma. En la Figura 16 se observa un fragmento de código de cómo la clase *CC_Principal* define un método *getInstanciaUnica* que garantiza la creación de un objeto de ese tipo una sola vez. Las restantes veces que se invoque esa operación retorna la instancia creada anteriormente.

```
private static CC_Principal instanciaUnica = null;

private CC_Principal(CI_Principal iguP) {

public static CC_Principal getInstanciaUnica(CI_Principal iguP) {
    if (instanciaUnica == null) {
        instanciaUnica = new CC_Principal(iguP);
    }
    return instanciaUnica;
}
```

Figura 16. Método *getInstanciaUnica* de la clase *CC_Principal*.

2.3.4. Diagramas de secuencia

A partir de los CU descritos y los diagramas de clase del diseño se realizaron los diagramas de secuencia (DS) para modelar el comportamiento del SE durante el proceso de diagnóstico, o durante la búsqueda de datos de alguna enfermedad. A través de los CU se indican cómo los actores interactúan con el sistema, pero es mediante los diagramas de secuencia que se hace una representación gráfica de los eventos que fluyen de los actores hacia el sistema. Esto permite mostrar tales eventos en una secuencia numerada, proporcionando así una idea más clara de cómo se ejecuta el proceso. De esta forma se logra una visión más detallada de lo que se debe codificar y mejora la siguiente etapa de implementación. En la Figura 17 se muestra el DS para el CU *Realizar diagnóstico.*

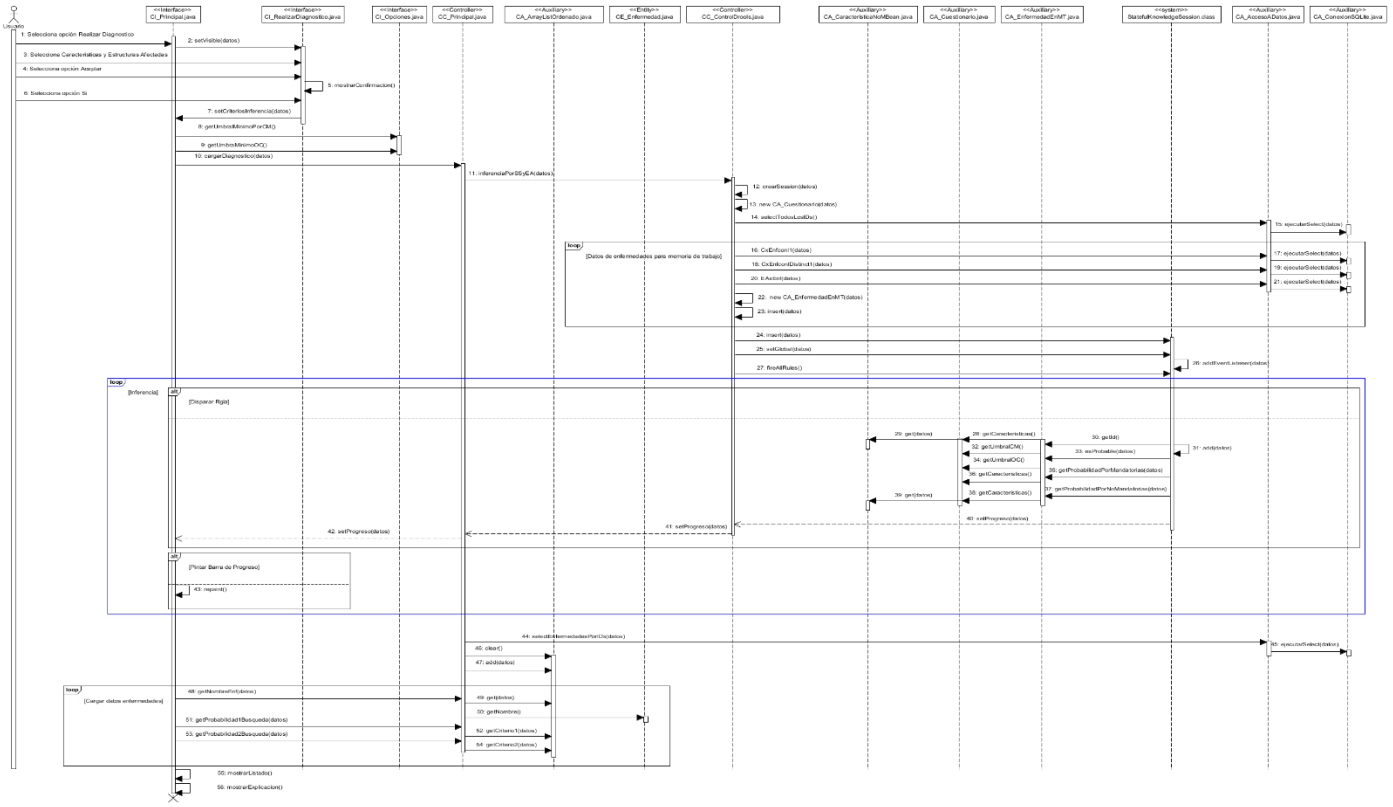


Figura 17. Diagrama de secuencia para el caso de uso *Realizar diagnóstico*.

En el diagrama anterior el actor *Usuario* selecciona en la interfaz Principal (*CI_Principal*) la opción *Realizar diagnóstico*, la cual crea un objeto de tipo *CI_RealizarDiagnostico*. En la clase *CI_RealizarDiagnostico* se muestran los signos y síntomas organizados por estructuras del cuerpo humano para que el actor pueda seleccionarlos para realizar el diagnóstico. Una vez seleccionados estos elementos, la clase *CI_RealizarDiagnostico* modifica en *CI_Principal* las variables que se utilizan para almacenar los criterios para realizar el proceso de inferencia. La clase *CI_Principal* entonces invoca los métodos necesarios de la clase *CI_Opciones* para obtener los valores de los umbrales (explicados en el DCD del mismo CU) para realizar el diagnóstico y ejecuta una llamada al método *cargarDiagnostico* de la clase controladora *CC_Principal,* enviándole los criterios de inferencia establecidos.

La clase *CC_Principal* invoca el método de la clase *CC_ControlDrools* para realizar la inferencia usando el subsistema Drools, enviándole como parámetros los criterios de inferencia, y tres listas vacías *(ids, nPM, nPO)* para que se almacene la información. La clase *CC_ControlDrools* crea un objeto (*session*) de tipo *StatefulKnowledgeSession* (clase del subsistema Drools) para acceder a las reglas de la BC, crea un objeto

(*c*) de tipo *CA_Cuestionario* con los criterios de inferencia y crea una lista (*list*) con los datos de todas las enfermedades en la base de datos; entonces el objeto *session* agrega a la memoria de trabajo el objeto *c* y los datos de *list* mediante el método *insert* y establece las variables globales mediante el método *setGlobal*. Se adiciona entonces un objeto de tipo *AgendaEventListener (evtL)* mediante el método *addEventListener* para monitorizar el proceso de inferencia y se disparan las reglas mediante el método *fireAllRules* del objeto *session*.

A partir de este punto por cada regla que se dispara se obtiene el cuestionario *c* en memoria de trabajo y la enfermedad con *id* igual al de la enfermedad que valora la regla (objeto *ob*). Luego se llama el método *esProbable* del objeto *ob,* si este retorna *true* quiere decir que es posible que el paciente padezca esa enfermedad. Entonces se invocan los métodos *getProbabilidadPorMandatorias* y *getProbabilidadPorNoMandatorias* del objeto *ob,* y se adicionan los resultados de esas llamadas en las variables *nPM, nPO;* en la variable *ids,* se adiciona el identificador de la enfermedad. Por último se retira de la memoria de trabajo el objeto *ob*, que contiene los datos de la enfermedad que se analizó. Al finalizar la ejecución de la regla el objeto *evtL* envía un mensaje a través de la clase *CC_ControlDrools,* a la clase *CC_Principal* y esta a su vez a la clase *CI_Principal,* para mostrar el progreso de la operación que se realiza.

Después que se ejecuta el proceso de inferencia (todas las reglas han sido analizadas), la clase *CC_Principal* obtiene de la base de datos, a través de un objeto de la clase *CA_AccesoADatos*, una lista de enfermedades (tipo *CE_Enfermedad*) y las organiza de acuerdo a los valores de las probabilidades calculadas en las reglas, contenidos en las variables *nPM* y *nPO.* Al finalizar esta operación el diagnóstico está cargado en memoria (una lista *le*, atributo de la clase *CC_Principal*). Después la clase *CI_Principal* obtiene los nombres de cada objeto tipo *CE_Enfermedad* y se le muestra al actor el listado de enfermedades y la probabilidad de que el paciente padezca cada una de ellas teniendo en cuenta cada una de sus características.

Para consultar la descripción de los restantes diagramas de secuencia debe remitirse al expediente de proyecto, específicamente a la plantilla *010215_Modelo_de_diseño_SEDEGEN.doc*.

2.3.5. Modelo de datos

Diagrama entidad-relación

En un diagrama entidad-relación tiene se reflejan las entidades que constituyen información del mundo real a nivel conceptual; en ellas se reflejan los objetos (o entidades persistentes) primarios que se procesan en un sistema y su composición, de los cuales se almacena información persistente, pero desde el punto de vista de cómo se reflejan estos en tablas en la base de datos. Para el SE SEDEGEN se realizó un modelo de este tipo para reflejar la estructura de la base de datos que contiene toda la información de las enfermedades genéticas que se maneja, como se observa en la Figura 18.

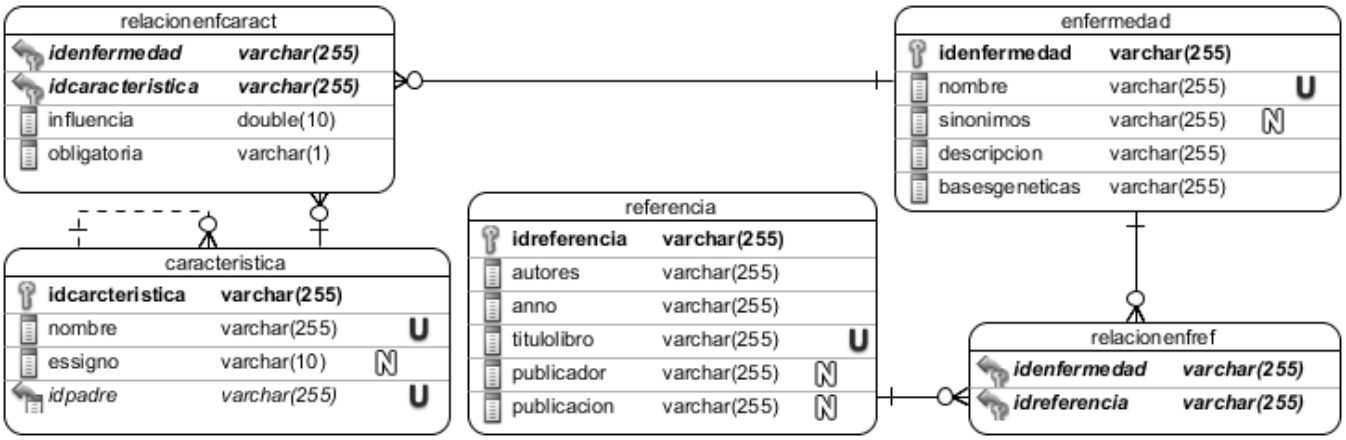


Figura 18. Diagrama entidad-relación del sistema SEDEGEN.

A continuación se realiza la descripción de las entidades más significativas:

Tabla 7. Descripción de la tabla enfermedad en la base de datos.

Nombre: enfermedad		
Descripción: Almacena los datos de las enfermedades genéticas.		
Atributo	**Tipo**	**Descripción**
idenfermedad	varchar(255)	Identificador de la entidad en la base de datos.
nombre	varchar(255)	Nombre de la enfermedad.
sinonimos	varchar(255)	Sinónimos de la enfermedad, otros nombres con los que se le conoce.
descripcion	varchar(255)	Descripción de forma general de la enfermedad.
basesgeneticas	varchar(255)	Bases genéticas de la enfermedad.

Tabla 8. Descripción de la tabla referencia en la base de datos.

Nombre: referencia		
Descripción: Almacena los datos de las referencias bibliográficas de las enfermedades genéticas.		
Atributo	**Tipo**	**Descripción**
idreferencia	varchar(255)	Identificador de la entidad en la base de datos.
autores	varchar(255)	Autores del libro.
titulo	varchar(255)	Título del libro.
anno	varchar(255)	Año de publicación del libro.
publicador	varchar(255)	Publicador del libro.
publicacion	varchar(255)	Publicación del libro.

Tabla 9. Descripción de la tabla característica en la base de datos.

Nombre: caracteristica		
Descripción: Almacena los nombres de los nomencladores (signos, síntomas y estructuras afectadas por las enfermedades genéticas).		
Atributo	**Tipo**	**Descripción**
idcaracteristica	varchar(255)	Identificador de la entidad en la base de datos.
nombre	varchar(255)	Nombre del nomenclador.
idpadre	varchar(255)	Identificador del nomenclador donde se localiza. Se debe poner 0 si es una estructura no contenida en otras (ej: sistema nervioso central).
essigno	varchar(10)	Define si la característica se refiere a un signo o a una estructura afectada o signo más general.

Conclusiones del capítulo

El planteamiento y análisis del problema existente en el CNGM para realizar diagnósticos de enfermedades genéticas, permitió una correcta identificación de los requisitos del SE. Esto propició la realización de un correcto diseño de los componentes y estructuras del sistema, donde se aplicaron patrones de arquitectura y diseño, mejorando la calidad de forma general de los artefactos generados: diagrama de casos de uso, diagramas de clases, diagramas de secuencia y modelo de datos. Mediante la generación de estos artefactos se logró comprender mejor el funcionamiento del sistema lo que posibilita realizar de forma efectiva la siguiente fase de implementación.

Capítulo 3: Implementación y prueba de un prototipo

En este capítulo se plasman los principales elementos del proceso de implementación del sistema, realizado con el objetivo de darle solución a los requisitos funcionales y se corresponden con las dos últimas fases que se emplean de la metodología: Implementación de un Prototipo y Prueba del Prototipo. Se describen los diagramas de componentes y los principales elementos del estándar de codificación que se emplea. También se muestra cómo se realizó la implementación de la base de conocimiento, así como la forma de expandirla desde el punto de vista de los desarrolladores. Además se presentan los resultados de las pruebas realizadas al software para comprobar la correcta implementación de cada una de las funcionalidades.

3.1. Diagrama de componentes

En un diagrama de componentes (DC) se muestran las partes modulares que encapsulan la implementación y se despliegan en un sistema, por ejemplo módulos y archivos. Además se reflejan las relaciones y dependencias entre las partes (componentes), así como con otras entidades, para dar cumplimiento a los requisitos. No es necesario en un diagrama reflejar todo el sistema, generalmente se divide por apartados. En el diagrama mostrado en la Figura 19, se muestran los componentes vinculados a la ejecución del CU *Realizar diagnóstico*.

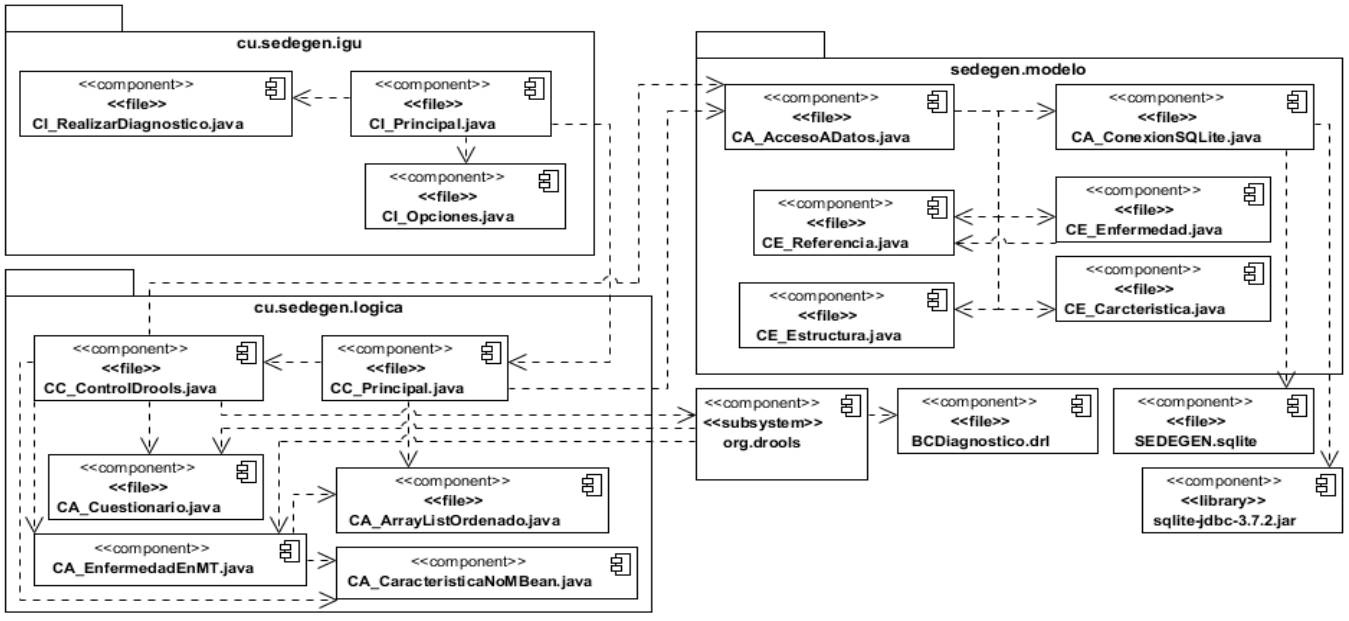


Figura 19. DC para el CU *Realizar diagnóstico*.

En la figura anterior se muestran los siguientes elementos organizados por capas:

Capa Presentación (cu.sedegen.igu):

- *CI_RealizarDiagnostico.java:* Contiene la clase *CI_RealizarDiagnostico*.
- *CI_Principal.java*: Contiene la clase *CI_Principal*.
- *CI_Opciones.java*: Contiene la clase *CI_Opciones*.

Capa Lógica (cu.sedegen.logica):

- *CC_ControlDrools.java*: Contiene la clase *CC_ControlDrools*.
- *CC_Principal.java*: Contiene la clase *CC_Principal*.
- *CA_Cuestionario.java:* Contiene la clase *CA_Cuestionario*.
- *CA_ArrayListOrdenado.java:* Contiene la clase *CA_ArrayListOrdenado*.
- *CA_EnfermedadEnMT.java:* Contiene la clase *CA_EnfermedadEnMT*.
- *CA_CaracteristicaNoMBean.java*: Contiene la clase *CA_CaracteristicaNoMBean*.

Capa Modelo (cu.sedegen.modelo):

- *CA_AccesoADatos.java*: Contiene la clase *CA_AccesoADatos*.
- *CA_ConexionSQLite.java*: Contiene la clase *CA_ConexionSQLite*.
- *CE_Referencia.java*: Contiene la clase *CE_Referencia*.
- *CE_Enfermedad.java:* Contiene la clase *CE_Enfermedad*.
- *CE_Estructura.java:* Contiene la clase *CE_Estructura*.
- *CE_Caracteristica.java*: Contiene la clase *CE_Caracteristica*.

Otros ficheros, librerías y subsistemas que intervienen:

- *BCDiagnostico.drl*: Contiene la BC del sistema. Contiene todas las reglas de producción.
- *Drools (org.drools):* Contiene todas las librerías y clases del subsistema Drools.
- *SEDEGEN.sqlite:* Fichero donde se almacena la base de datos con toda la información de las enfermedades genéticas.

- *sqlite-jdbc-3.7.2*: Librería usada para establecer la conexión al fichero *SEDEGEN.sqlite*.

3.2. Estándar de codificación

Un estándar de codificación establece las pautas y normas a seguir por los desarrolladores de un sistema para que su código fuente sea escrito en un formato unificado, logrando mayor organización y efectividad durante la fase de implementación. En la codificación de SEDEGEN se utilizaron los estándares establecidos para la codificación en Java, reflejados en (Hommel, 1999) de acuerdo a las particularidades del sistema. Algunos de los aspectos importantes a tener en cuenta son:

- Evitar las líneas de más de 80 caracteres.
- Siempre que sea posible inicializar las variables locales donde se declaran.
- Poner las declaraciones solo al principio de los bloques.
- Cada línea debe contener como máximo una sentencia simple (ej.: *c++).*
- El prefijo del nombre de un paquete se escribe siempre con letras ASCII en minúsculas, y debe ser uno de los nombres de dominio de alto nivel (com, gov) o uno de los códigos ingleses de dos letras que identifican a cada país (cu, es, en). Los subsecuentes componentes del nombre del paquete varían de acuerdo a las convenciones de nombres internas de cada organización.
- Los nombres de las clases e interfaces deben ser sustantivos, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas.
- Los nombres de los métodos deben ser verbos, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que los forman en mayúscula.

En la Figura 20 se muestra un fragmento del código en lenguaje Java implementado aplicando el estándar de codificación.

```
public void cargarDiagnostico(ArrayList<String> signosYsintomas,
        ArrayList<String> EA, double umbralM, double umbralOC)
        throws SQLException, ClassNotFoundException, Exception {

    ArrayList<Double> auxPM = new ArrayList<Double>();
    ArrayList<Double> auxPO = new ArrayList<Double>();
    ArrayList<String> idsEnf = new ArrayList<String>();

    ccControlDrools.realizarInferencia(signosYsintomas, EA, idsEnf, auxPM,
            auxPO, umbralM, umbralOC);

    CA_AccesoADatos accesoD = new CA_AccesoADatos();
    ArrayList<CE_Enfermedad> listaE = accesoD
            .selectEnfermedadesPorIDs(idsEnf);

    setProgreso(95);
    le.clear();

    int cant = listaE.size();
    for (int j = 0; j < cant; j++) {
        int pos = idsEnf.indexOf(listaE.get(j).getIdBD());
        le.add(listaE.get(j), auxPM.get(pos), auxPO.get(pos));
    }
    setProgreso(100);
}
```

Figura 20. Ejemplo de código fuente aplicando los estándares de codificación.

3.3. Base de Conocimiento

El conocimiento obtenido del experto se representó en forma de reglas de producción mediante el lenguaje definido por Drools; estas fueron almacenadas en un archivo, de forma tal que el sistema las utilice en tiempo de ejecución. Cada regla posee los criterios suficientes para determinar la probabilidad de que un paciente posea determinada enfermedad; por tanto se elaboró una regla para cada enfermedad, siendo en total 285 reglas de producción. El proceso de inferencia se realiza teniendo en cuenta el nivel de coincidencia que tengan los síntomas del paciente con los síntomas de la enfermedad que se analiza en la regla. El resultado final es un conjunto de enfermedades ordenadas por la probabilidad de que un paciente padezca cada una de ellas. Precisamente una de las características de muchos sistemas expertos y del que se aborda en el presente trabajo en particular, es que no dan un único resultado de forma definitiva, sino que propone una serie de posibles diagnósticos para que el usuario (en este caso, un genetista) los valore

y determine finalmente cuál es el más apropiado. Por este motivo el sistema se puede clasificar como un sistema de apoyo en la toma de decisiones.

Partiendo de la estructura de la forma de representación mediante reglas de producción, estas se concibieron de la siguiente forma: SI <es probable padecimiento de enfermedad X> ENTONCES <diagnosticar enfermedad X>. En la Figura 21 se muestra una regla en el código definido por Drools para determinar si el paciente puede padecer la enfermedad "*Síndrome de la trisomía 18*". En las reglas se usan las palabras reservadas *when* para definir los antecedentes o condiciones de la regla y la palabra *then* para definir las acciones o los consecuentes de la misma.

```
rule "R2: SINDROME DE LA TRISOMIA 18"
    when
        $c:= CA_Cuestionario()

        $o:= CA_EnfermedadEnMT( getId() == "2", esProbable($c) )
    then
        idsGL.add($o.getId());
        nPMGL.add($o.getProbabilidadPorMandatorias($c));
        nPOGL.add($o.getProbabilidadPorNoMandatorias($c));
        retract($o);
end
```

Figura 21. Regla definida en lenguaje Drools.

La elaboración de las reglas de producción de esta forma garantiza la posibilidad de una rápida expansión de la BC sin necesidad de escribir más de una vez los datos de la nueva enfermedad. Solo incorporando la información en la base de datos una vez, las reglas los obtienen dinámicamente en tiempo de ejecución, así como la representación visual de los mismos en la interfaz gráfica de usuario (IGU). De esta forma se brinda la posibilidad de incorporar mayor conocimiento experto de forma sencilla y rápida. Para elaborar una regla que determine la probabilidad de que el paciente padezca una nueva enfermedad que no se tenía en cuenta antes en la BC, se deben seguir los pasos mostrados a continuación:

- Adicionar en la base de datos, en la tabla *enfermedad*, los datos de la nueva enfermedad.
- Adicionar en la tabla *característica* las estructuras afectadas, los signos y síntomas que provoca la enfermedad y sus relaciones en la tabla *relacionenfcaract* con la(s) enfermedad(es) que los provocan.
- Escribir en la BC (fichero *BCDiagnostico.drl*) la estructura de la regla como se define en Drools: palabras reservadas *rule, when, then, end.*

- Escribir luego de la palabra *rule* una cadena de texto entre comillas que será el identificador de la regla.
- Escribir las 2 líneas de código mostradas en la Figura 22 entre las palabras reservadas *when* y *then*, cambiando en la parte ***getId() == "2",*** el 2 por el identificador de la enfermedad en la tabla *enfermedad* de la base de datos.
- Adicionar alguna regla auxiliar si fuese necesario para realizar control de coherencia o tener en cuenta algún otro elemento para realizar el diagnóstico de dicha enfermedad que no sean las características y las estructuras afectadas que provoca.

3.4. Prueba del prototipo

Las pruebas son un conjunto de actividades que se planean con anticipación y se realizan de manera sistémica para evaluar la calidad de un producto (software, componente) y descubrir errores en él (Pressman, 2010). De esto se deduce que mediante las pruebas se pueden encontrar fallos en el comportamiento de un sistema y errores, aunque no se puede demostrar la ausencia de ellos. No obstante, son una importante herramienta para los desarrolladores de software para comprobar el correcto funcionamiento del sistema así como del cumplimiento de los requisitos.

Para probar el sistema SEDEGEN, durante la fase de desarrollo, se realizaron *pruebas de unidad*. Estas pruebas, contextualizado en la POO, se utilizan para evaluar las unidades más pequeñas del software: clases, objetos y funciones. Para realizarlas se empleó el IDE Eclipse, donde se crearon casos de prueba *JUnit* (*JUnit Test Case*), donde se verificó el correcto funcionamiento del procedimiento que responde al CU crítico *Realizar diagnóstico*, así como los procedimientos para realizar las búsquedas de datos de las enfermedades, mediante las clases *CC_ControlDroolsTest* y *CC_PrincipalTest* respectivamente. En ambos casos se aplicaron valores límites y dentro de los límites de entrada a las funciones y el sistema respondió satisfactoriamente. En la Figura 25 (Anexo 2) se muestra una de las variantes de código aplicado para realizar la prueba *JUnit* para comprobar el funcionamiento del método para realizar la inferencia, de la clase *CC_ControlDrools*.

Además se realizaron *pruebas de funcionalidad* aplicando el método de *caja negra*. Las pruebas de caja negra se basan en la realización de pruebas sobre la interfaz del programa, evaluando las entradas y salidas del mismo, de esta forma se comprueba el cumplimiento de todos los requisitos del sistema. Para su aplicación se elaboraron casos de prueba a partir de las funcionalidades descritas en los casos de uso del

sistema y de la descripción de los mismos como apoyo para las revisiones. En cada caso de prueba se refleja la especificación de un caso de uso, dividido en secciones y escenarios, donde se detallan las funcionalidades descritas en él y se describen las variables utilizadas. En la aplicación de este método se empleó la técnica *Partición equivalente* para elaborar los casos de prueba, donde los datos de entrada se dividieron en clases de equivalencia[3]. De esta forma se garantiza que para una condición de entrada se evalúen todos los casos posibles. En la Tabla 10 se describe el caso de prueba para el CU *Realizar diagnóstico*, para la sección con el mismo nombre y en la Tabla 11 se realiza la descripción de las variables.

Tabla 10. Caso de prueba *Realizar diagnóstico*: Sección *Realizar diagnóstico*.

Escenario	**Descripción**	**Signos**	**Respuesta del sistema**	**Flujo central**
EC 1.1 El actor selecciona los signos, síntomas y/o estructuras.	En este escenario el actor selecciona los signos, síntomas y/o estructuras afectadas y selecciona la opción *Aceptar*.	V (Hipotonía, Facies plana, Fisuras palpebrales inclinadas, Orejas pequeñas)	El sistema realiza el diagnóstico y muestra la enfermedad "Síndrome de down" como la más probable.	1- Seleccionar los signos, síntomas y/o estructuras en el árbol donde se muestran. 2- Adicionarlos a la lista de elementos seleccionados. 3- Seleccionar la opción *Aceptar*.
EC 1.2 El actor no selecciona los signos, síntomas y/o estructuras.	En este escenario el actor no selecciona ningún signo, síntoma o estructuras y selecciona la opción *Aceptar*. El sistema verifica que el actor haya seleccionado al menos un signo, síntoma o estructura.	NA	El sistema muestra un aviso indicando que debe seleccionar al menos un signo, síntoma o estructura afectada.	1- Seleccionar la opción *Aceptar* sin haber seleccionado ningún signos, síntoma o estructura del árbol

[3] Una clase de equivalencia representa un conjunto de estados válidos y no válidos para las condiciones de entrada.

Tabla 11. Descripción de las variables del caso de prueba *Realizar diagnóstico*.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Signos	Lista	No	Debe seleccionar al menos un signo, síntoma o estructura afectada.

Después de realizadas las pruebas funcionales, los errores encontrados fueron redactados como no conformidades para ser solucionados por el equipo de desarrollo. En total se realizaron tres iteraciones y se encontraron nueve no conformidades, las cuales fueron resueltas en cada iteración correspondiente, como se muestra en la Figura 22. En la Tabla 12 se muestran algunas de las no conformidades encontradas.

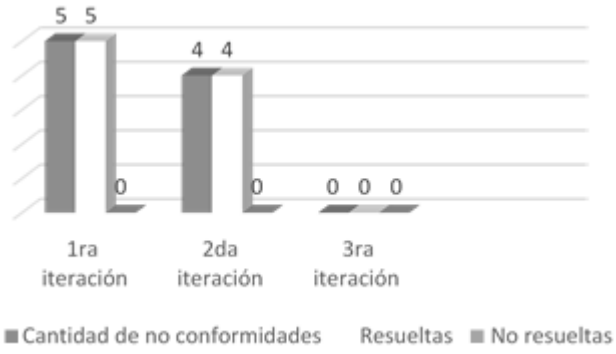


Figura 22. Gráfica de no conformidades encontradas en cada iteración.

Tabla 12. Ejemplo de no conformidades encontradas.

Elemento	**No.**	**No Conformidad**	**Ubicación**	**Etapa de detección**	**Clasificación**
Aplicación	1	El campo “Nombre de enfermedad” acepta caracteres extraños (‘ y %).	Realizar búsqueda de enfermedades por nombre.	Pruebas de funcionalidad.	Significativa
Aplicación	2	El campo “Año de publicación del libro” acepta valores no numéricos, caracteres extraños y valores mayores de 4 caracteres.	Realizar búsqueda de enfermedades por referencias bibliográficas.	Pruebas de funcionalidad	Significativa

Aplicación	3	El campo “Título del libro” acepta caracteres extraños.	Realizar búsqueda de enfermedades por referencias bibliográficas.	Pruebas de funcionalidad	Significativa
Aplicación	4	El campo “Autor del libro” acepta caracteres extraños y números.	Realizar búsqueda de enfermedades por referencias bibliográficas.	Pruebas de funcionalidad	Significativa

Además para probar el rendimiento del sistema se realizaron pruebas donde se midió el tiempo de respuesta al ejecutar alguna búsqueda o realizar algún diagnóstico. En dependencia de las características de las computadoras donde se probó, en cuanto a cantidad de memoria RAM y a prestaciones del microprocesador, se obtuvieron distintos resultados, lo que ayudó a comprobar los requisitos mínimos de hardware que debe poseer la PC donde se instale el sistema. En la Tabla 13 se muestran los resultados obtenidos al ejecutar diagnósticos con el máximo número de criterios posibles (786 características para un paciente), lo que se considera que es una cantidad mucho mayor a las que se especificarán en la realidad, garantizando así una cota máxima de tiempo. Las búsquedas se ejecutaron de forma normal, probando las distintas configuraciones posibles para realizarlas y los tiempos de respuesta siempre fueron menores de 1 segundo.

Tabla 13. Comparación de los tiempos de respuesta del sistema.

Memoria RAM	**Microprocesador**	**Tiempo de respuesta en segundos de los diagnósticos**
1 GB	Intel(R) Celeron(R) CPU E3400 2.60 GHz	12
1 GB	Intel(R) Pentium(R) 4	24
1 GB	Pentium(R) Dual-Core CPU E5300 2.60 GHz	10
1 GB	AMD A8 4500M Quad Core 1.90 GHz (en una máquina virtual)	24
2 GB	Intel(R) Pentium(R) D CPU 3.00 GHz	17
2 GB	Intel(R) Atom(TM) CPU D525 1.80GHz	36

De la anterior tabla se deduce que, efectivamente, el sistema funcionará brindando tiempos de respuesta menores a 24 segundos para los diagnósticos y menores de 4 segundos para las búsquedas, si la PC donde se instale tiene un microprocesador Intel(R) Pentium(R) 4 o superior con 1GB de memoria RAM como

mínimo. En condiciones de hardware inferiores a las mencionadas el sistema pudiera funcionar correctamente, pero los tiempos de respuestas podrían ser mayores a los especificados.

3.5. Interfaces del sistema

Como resultado del proceso de implementación se muestran algunas interfaces del sistema. En la Figura 23 se presenta la interfaz para realizar los diagnósticos, en la misma se seleccionan los síntomas, signos y estructuras afectadas que presenta el paciente para que el sistema brinde posibles enfermedades a partir de los elementos seleccionados.

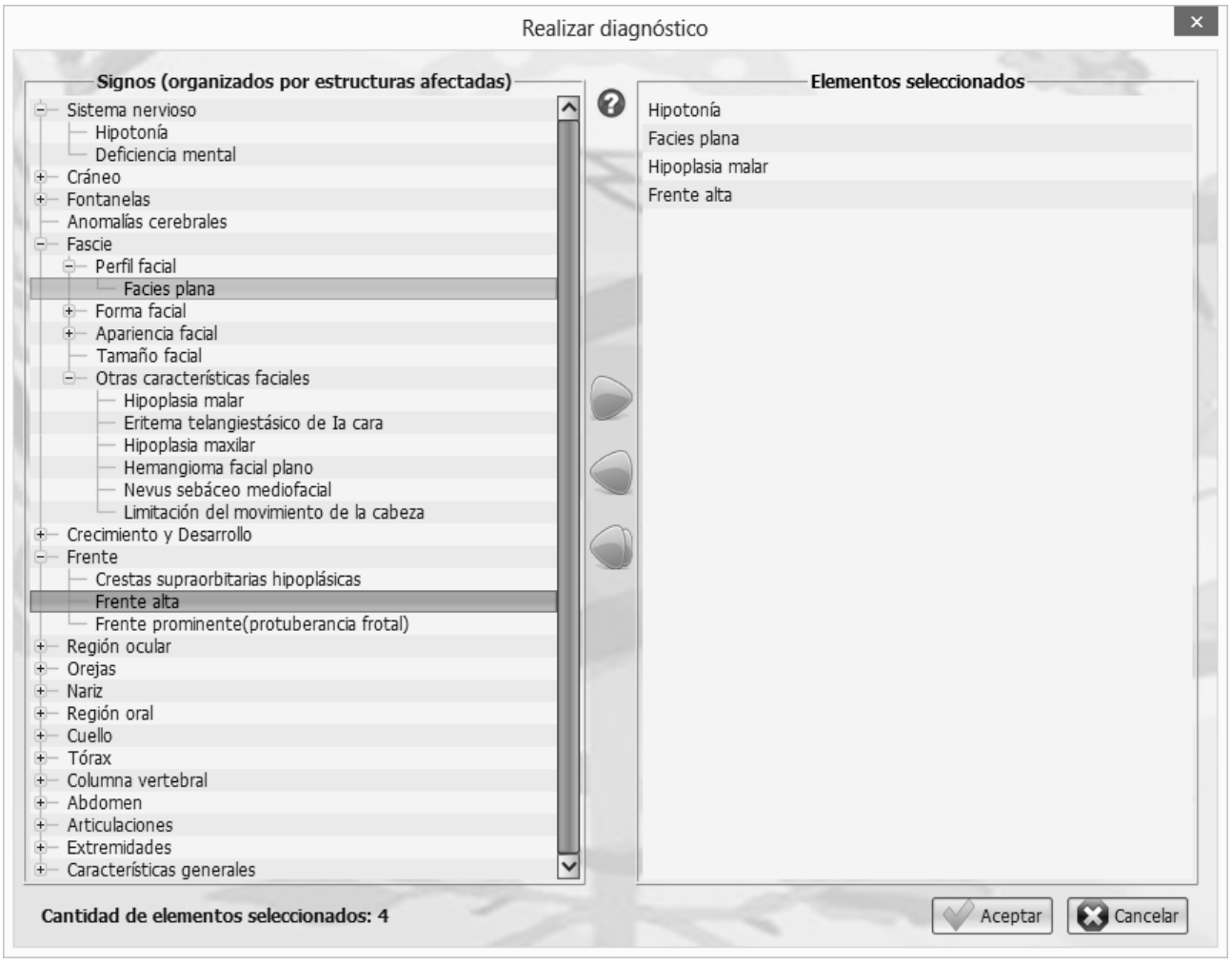


Figura 23. Interfaz para seleccionar los signos que presenta el paciente para realizar el diagnóstico.

Luego de ejecutar el proceso de inferencia, el sistema muestra en la interfaz principal la explicación de los resultados obtenidos, brindando una enfermedad como posible diagnóstico y otras posibles enfermedades a valorar, como se muestra en la Figura 24.

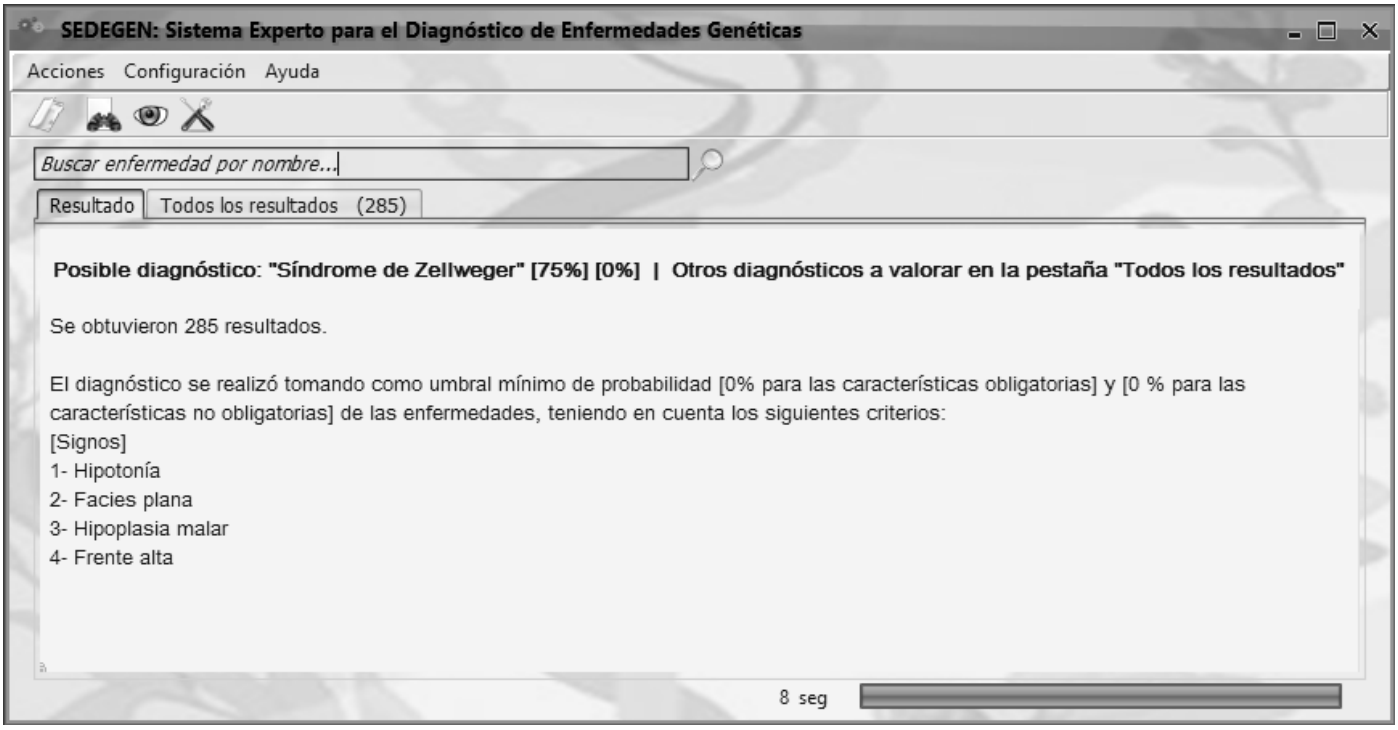


Figura 24. Interfaz principal. Se muestra la explicación del diagnóstico realizado.

Conclusiones del capítulo

Partiendo del diseño del sistema, se hizo la representación formal del conocimiento experto mediante reglas de producción y se implementaron las funcionalidades para realizar diagnósticos y búsquedas de datos de enfermedades genéticas usando el lenguaje Java, las cuales fueron validadas mediante pruebas de software. El uso de estándares de codificación en la generación del código fuente posibilita una mayor facilidad de entendimiento de la implementación de las funcionalidades y facilita el futuro mantenimiento y soporte del sistema. Además mediante la forma de elaboración de las reglas de la BC se garantiza una rápida expansión de la misma, ya que solo se deben escribir una vez los datos de las enfermedades en la base de datos y elaborar la(s) regla(s) correspondiente(s) para que estos sean mostrados en la IGU y analizados en las reglas.

Conclusiones generales

Después de realizada la investigación y de haberse construido el Sistema Experto, transitando por todas las fases desde el análisis hasta las pruebas, se puede arribar a las siguientes conclusiones:

- A través del estudio de los fundamentos teóricos de los sistemas expertos se logró seleccionar la metodología de desarrollo, las herramientas y tecnologías necesarias para la construcción del Sistema Experto para el Diagnóstico de Enfermedades Genéticas.
- El planteamiento y análisis del problema en el diagnóstico de enfermedades genéticas del CNGM, permitió una correcta identificación de los requisitos del sistema.
- A través del diseño del sistema se logró definir su organización lógica, haciendo uso de patrones de diseño.
- La adquisición del conocimiento experto, obtenido del especialista del CNGM, permitió la elaboración de reglas de producción e implementación de funcionalidades que posibilitan la búsqueda de información y diagnóstico de enfermedades genéticas a partir de los datos de los pacientes.
- La aplicación de pruebas de software permitió verificar el funcionamiento del prototipo desarrollado, lográndose corregir todos los errores encontrados.
- De forma general, el sistema SEDEGEN constituye una valiosa herramienta para los especialistas del CNGM, pues además de consultar la información referente a las enfermedades genéticas de forma automatizada, cuentan con un sistema capaz de brindarle una opinión de apoyo para los diagnósticos que realizan a los pacientes.

Recomendaciones

- Expandir la BC del sistema para que sea capaz de diagnosticar una mayor cantidad de enfermedades genéticas, incorporando la nueva información en la base de datos e implementando las reglas de producción correspondientes.
- Implementar un subsistema de aprendizaje para incorporar conocimiento que no se tenga almacenado hasta el momento.
- Implementar las funcionalidades necesarias para realizar la validación de un diagnóstico, es decir, a partir de un diagnóstico determinado por el especialista, que el sistema determine cuán probable puede ser que el paciente padezca la enfermedad diagnosticada. Para esto se debe usar el encadenamiento hacia atrás brindado por el Sistema de Gestión de Reglas del Negocio Drools.

Referencias bibliográficas

Buschmann, Frank, y otros. 1996. *PATTERN-ORIENTED SOFTWARE ARCHITECTURE. Vol 1.* [PDF] New York : John Wiley & Sons, 1996. ISBN 0 471 95889 7.

Cabrera B, Layda de los Ángeles y Torres A, José M. 2012. *Sistema experto para el tratamiento médico de pacientes con Hipertensión Arterial y Diabetes Mellitus.* [PDF] La Habana : s.n., 2012.

Castillo, Enrique, Gutiérrez, José M y Hadi, Ali S. 1996. *Sistemas Expertos y Modelos de Redes Probabilísticas.* [PDF] 1996. ISBN: 9788460093954.

Cuadrado R, Santiago, y otros. 2011. Sistema experto basado en casos para el diagnóstico de la hipertensión arterial. [En línea] 2011. [Citado el: 1 de Diciembre de 2013.] http://aprendeenlinea.udea.edu.co/revistas/index.php/ingenieria/article/view/13748.

Ferrer M, Ernesto y García S, Jorge A. 2008. *Desarrollo del sistema SEDIM-SV utilizando la metodología Weiss-Kulikowski.* [PDF] La Habana : s.n., 2008.

Friedman-Hill, Ernest. 2013. Jess, The Rule Engine for JavaTM Platform. [En línea] 19 de Sep de 2013. [Citado el: 25 de Oct de 2013.] http://www.jessrules.com/jess/index.shtml.

Gálvez L, Daniel. 2006. *Sistemas Basados en el Conocimiento.* [PDF] Santa Clara : Universidad Central "Martha Abreu" de las Villas, 2006.

Gamma, Erich, y otros. 1994. *Design Patterns. Elements of Reusable Object-Oriented Software.* [PDF] Zurich, Sydney, Urbana, Hawthorne : KevinZhang, 1994. ISBN-10: 0201633612, ISBN-13: 978-0201633610.

Garlan, David y Shaw, Mary. 1994. *An Introduction to Software Architecture.* [PDF] Pittsburg : World Scientific Publishing Company, 1994.

Gosling, James, y otros. 2013. *The Java Language Specification. Java SE 7 Edition.* [PDF] California : Oracle America, Inc., 2013.

Gutiérrez R, Marianela y Bedoya R, Jorge. 2010. *Sistema experto para el diagnóstico médico de las enfermedades genéticas con dismorfias(SEGEDIS).* [PDF] La Habana : s.n., 2010.

Henao C, Mónica. 2001. *CommonKADS-RT: Una Metodología para el Desarrollo de Sistemas Basados en el Conocimiento de Tiempo Real.* [PDF] Valencia : s.n., 2001.

Hommel, Scott. 1999. *Java Code Conventions.* [PDF] 1999.

Jones, Kennet L. 2007. *SMITH. Patrones reconocibles de malformaciones humanas. 6ta Edición.* Barcelona : ELSEVIER SAUNDERS, 2007.

Jova R, Jorge R. 2012. *Metodología CommonKADS en el desarrollo de sistemas expertos.* [PDF] La Habana : Universidad de las Ciencias Informáticas, 2012.

Kruchten, Philippe. 1995. *Architectural Blueprints - The "4+1" View.* [PDF] s.l. : IEEE, 1995.

Larman, Craig. 1999. *UML Y PATRONES. Introducción al análisis y diseño orientado a objetos.* [PDF] México : PRENTICE HALL, 1999. ISBN: 970-17-0261-1.

Marcheco T, Beatriz. 2008. *Revista Cubana de Genética Comunitaria.* 2008. Vol. II.

Microsoft. 2014. Microsoft. *Microsoft.* [En línea] Marzo de 2014. [Citado el: 2 de Marzo de 2014.] http://msdn.microsoft.com/en-us/library/ee658098.aspx.

Ojeda, Norma Elena de León. 2013. Genética Médica, Infomed. *Genética Médica, Infomed.* [En línea] 2013. [Citado el: 23 de Noviembre de 2013.] http://www.sld.cu/sitios/genetica/verpost.php?pagina=1&blog=http://articulos.sld.cu/genetica&post_id=391&c=2987&tipo=2&idblog=141&p=1&n=de.

Palma, J. T., y otros. 2000. *Ingeniería del Conocimiento. De la Extracción al Modelado de Conocimiento.* [PDF] 2000. ISSN: 1988-3064.

Peña A, Alejandro. 2006. *Sistemas basados en Conocimiento: Una Base para su Concepción y Desarrollo.* [PDF] Revillagigedo : Dirección de Publicaciones Instituto Politécnico Nacional, 2006. ISBN: 970-94797-4-1.

Pressman, Roger S. 2010. *Software Engineering: A Practitioner´s Approach, Seventh Edition.* [PDF] New York : McGrawHill, 2010. ISBN: 978-0-07-337597-7.

RAmEx Ars Medica, Inc. 2002. RAmEX Ars Medica. *RAmEX Ars Medica.* [En línea] Oxford University Press, 2002. [Citado el: 6 de Diciembre de 2013.] http://www.ramex.com/title.asp?id=2937.

Rich, Elaine y Knight, Kevin. 2000. *Inteligencia Artificial, Segunda Edifición.* [PDF] Madrid : Mc-GRAW-HILL, 2000. ISBN: 0-07-052263-4; ISBN: 84-481-1858-8.

Rumbaugh, James, Jacobson, Ivar y Booch, Grady. 1998. *El Lenguaje Unificado de Modelado. Manual de Referencia.* [PDF] California : Addison Wesley, 1998.

Rusell, Stuar y Norvig, Peter. 2010. *Artificial Intelligence. A Modern Approach. Third Edition.* [PDF] New Jersey : Prentice Hall, 2010. ISBN-13: 978-0-13-604259-4; ISBN-10: 0-13-604259-7.

Serrano S, Saralys y Placencia B, Alejandro. 2013. *Sistema experto basado en reglas para el apoyo al diagnóstico de la fragilidad en el adulto mayor.* [PDF] La Habana : s.n., 2013.

Sommerville, Ian. 2005. *Ingeniería del Software Séptima Edición.* [PDF] Madrid : PEARSON EDUCACIÓN, S.A, 2005. ISBN: 84-7829-074-5.

—. 2007. *Software Engineering (Eighth Edition).* [PDF] Hong Kong, Macau, Taiwan : Adison-Wesley Publishers, 2007. ISBN 13: 978-0-321-31379-9; ISBN 10: 0321-31379-8; ISBN 7-111-19770-4.

SourceForge. 2008. SourceForge. [En línea] 24 de January de 2008. http://clipsrules.sourceforge.net/WhatIsCLIPS.html.

—. 2013. SourceForge. [En línea] 2013. http://sourceforge.net/projects/jeops/.

SQLite. 2013. SQLite. [En línea] Diciembre de 2013. http://www.sqlite.org/.

Tamayo F, Martha L, Bernal V, Jaime y Latting, María C. 1996. *ASPECTOS GENÉTICOS BÁSICOS Y DE DISMORFOLOGÍA. Aplicaciones prácticas en medicina.* Bogotá : s.n., 1996.

The Eclipse Foundation. 2013. Eclipse - The Eclipse Foundation. [En línea] 6 de Diciembre de 2013. http://www.eclipse.org/.

The JBoss Team. 2013. JBoss.org. *JBoss.org.* [En línea] Diciembre de 2013. http://docs.jboss.org/drools/release/5.5.0.CR1/drools-expert-docs/html_single/#d0e26.

Visual Paradigm. 2013. Visual Paradigm. *Visual Paradigm.* [En línea] 2013. http://www.visual-paradigm.com/.

Anexos

Anexo 1. Preguntas realizadas como parte de las entrevistas.

- Para la obtención de requisitos del sistema.
    1. ¿Qué procedimiento sigue el especialista para realizar el diagnóstico de un paciente?
    2. ¿Qué criterios son usados por los especialistas para realizar la búsqueda de los datos de determinada enfermedad? De ellos, ¿cuáles son los más empleados?
    3. ¿Puede combinarse más de un criterio para realizar la búsqueda de los datos de una enfermedad?
    4. ¿Puede realizarse el diagnóstico de un paciente sin definir específicamente los síntomas de las enfermedades y solo especificar las estructuras del cuerpo afectadas?
    5. ¿Qué tiempo (estimado) demora un especialista en definir un diagnóstico en función de las búsquedas de información que realiza para apoyarse?
    6. ¿Cómo debe presentar la información técnica el sistema para un mejor entendimiento de los especialistas?
- Para la obtención del conocimiento del SE:
    1. ¿Cuáles son las principales fuentes de obtención de información sobre las enfermedades genéticas que usan los especialistas del CNGM como apoyo?
    2. ¿Son exhaustivas las fuentes de información referente a las enfermedades genéticas consultadas por los especialistas?
    3. ¿Existe alguna prioridad en la valoración de los criterios tenidos en cuenta para realizar un diagnóstico? De ser así, ¿cómo se debe evaluar?
    4. ¿Para que una enfermedad sea diagnosticada a un paciente, este debe presentar todos los síntomas de la misma?
    5. ¿Cómo definir qué enfermedad es más probable que padezca un paciente a partir de los criterios valorados?

Anexo 2. Código de prueba *JUnit* para comprobar el método *realizarInferencia de la clase CC_ControlDrools*

```
@Test
public void testCC_Drools_MetodoInferencia() throws Exception {

    CA_AccesoADatos ad = new CA_AccesoADatos();

    for (int i = 0; i < 200; i++) {
        aux = ad.selectNomencladorxIDPadre(String.valueOf(i));
        for (int j = 0; j < aux.size(); j++) {
            caract.add(aux.get(j).toString());
        }

        aux = ad.selectNomencladorxIDPadre(String.valueOf(i));
        for (int j = 0; j < aux.size(); j++) {
            EA.add(aux.get(j).toString());
        }
    }

    cd.realizarInferencia(caract, EA, idsEnf, nPM, nPO, 0.0, 0.0, 50);

    String aux2 = "|";
    for (int i = 0; i < idsEnf.size(); i++) {
        aux2 += idsEnf.get(i) + "|";
    }
    System.out.println(aux2);
}
```

Figura 25. Código de prueba JUnit.

Glosario de términos

API: Interfaz de Programación de Aplicaciones (del inglés Applications Programming Interface).

Diagnosis: Diagnóstico.

Disrupción: Ruptura, cambio significativo.

Jackrabbit: Es una implementación completa de la API “Content Repository” para la Tecnología Java (JCR, Java Content Repository).

JSR-94: Especificación del lenguaje Java que define una API para los motores de inferencia permitiendo acceder al mismo desde la plataforma Java en tiempo de ejecución.

Modus Ponens: Regla de inferencia donde se examina la premisa (antecedente) de la regla, y si es cierta, la conclusión (consecuente) pasa a formar parte del conocimiento (memoria de trabajo).

Morbimortalidad: Muerte causada por enfermedades en las poblaciones en determinados espacios y tiempos.

Servlets: Clase en el lenguaje de programación Java, utilizada para ampliar las capacidades de un servidor.

SQL92: Definición de un lenguaje estándar para Sistemas Gestores de Base de Datos Relacionales adoptado por el Instituto de Estándares Nacional Americano, ANSI (del inglés American National Standards Institute) y la Organización de Estándares Internacional, ISO (International Standards Organizations).