Universidad de las Ciencias Informáticas
Facultad 5

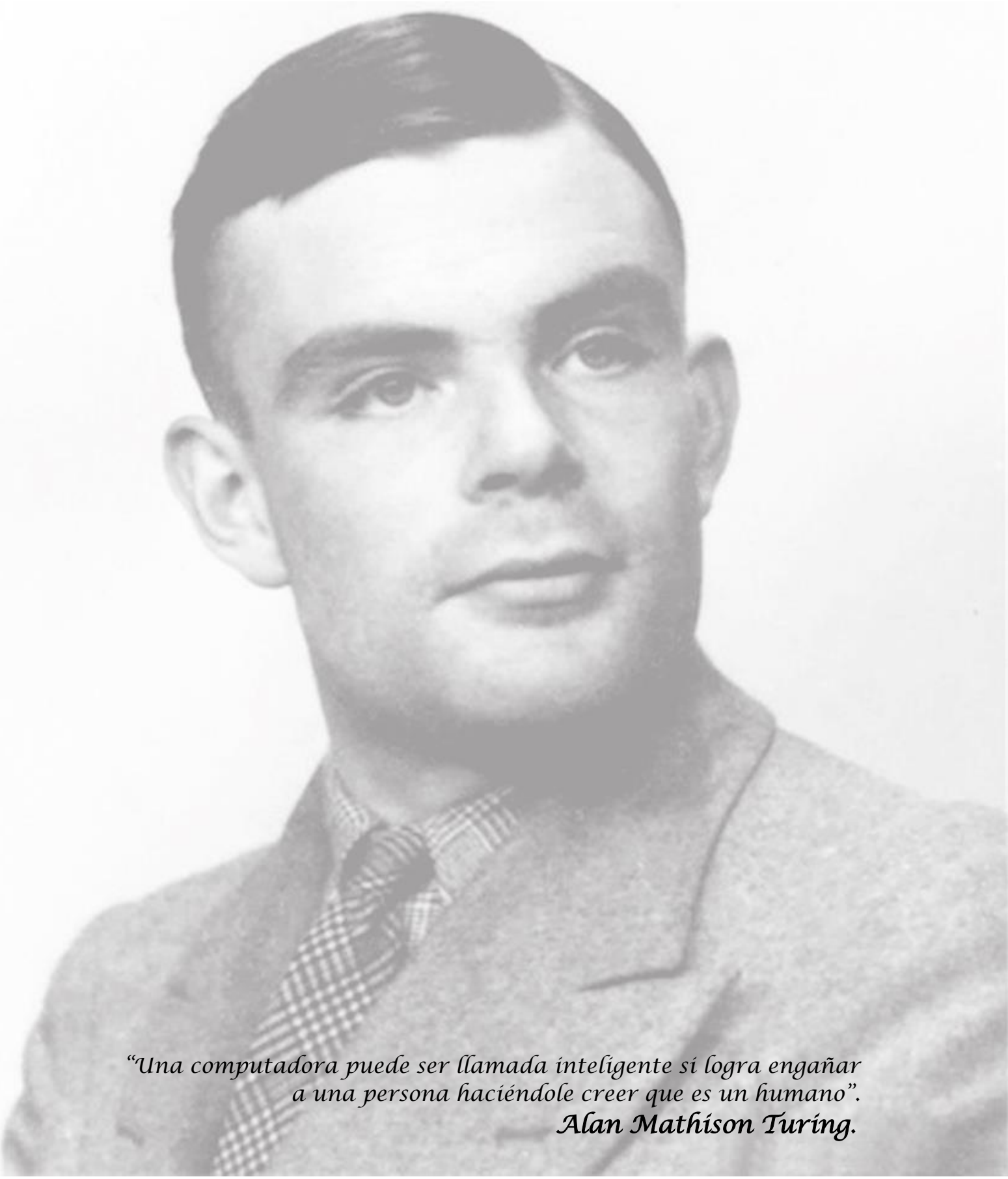
Sistema inteligente para la resolución práctica de problemas de Inteligencia Artificial utilizando la programación lógica.

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autor(es): Arletis Velázquez Ramírez.
Leduan Bárbaro Rosell Acosta.

Tutor: MSc. Yuniesky Coca Bergolla.
Co-tutor: Ing. Orlando Cruz Torres.

"Año 58 de la Revolución"
La Habana, Cuba
Junio 2016



"Una computadora puede ser llamada inteligente si logra engañar a una persona haciéndole creer que es un humano".

Alan Mathison Turing.

Declaración de autoría

Declaramos ser los únicos 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 _____.

_____ _____

Leduan Bárbaro Rosell Acosta.

Autor

Arletis Velázquez Ramírez.

Autor

_____ _____

MSc. Yuniesky Coca Bergolla.

Tutor

Ing. Orlando Cruz Torres.

Co-Tutor

Datos de contacto

Tutor:

MSc. Yuniesky Coca Bergolla.

Email: ycoca@uci.cu

Graduado de Licenciatura en Ciencias de la Computación en la Universidad Central de Las Villas (UCLV) 2003. Ha trabajado desde estudiante en temas relacionados con la inteligencia artificial (IA). Defendió su maestría en Informática Aplicada en la Universidad de las Ciencias Informáticas (UCI) en 2007. Ha tutorado más de 15 tesis de grado relacionadas con estos temas. Actualmente se desempeña como jefe de departamento de técnicas de programación de la facultad 5 y Jefe de disciplina de IA en la UCI. Además es árbitro de la Revista Cubana de las Ciencias Informáticas (RCCI).

Co-tutor

Ing. Orlando Cruz Torres

Email: orlandoct@uci.cu

Graduado de Ingeniería en Ciencias Informáticas (UCI) 2015. Su tesis de grado estuvo estrechamente relacionada con la Inteligencia Artificial. Es profesor del departamento de Ingeniería de Software de la facultad 5 de la Universidad de las Ciencias Informáticas.

Dedicatoria Arletis

A mi madre:

La única persona verdaderamente incondicional en mi vida. Porque su amor, su paciencia y su ternura me han dado fuerzas para ser la persona que soy hoy.

A mis abuelos Beysi y Eugenio:

Por sus enseñanzas, sus consejos, sus mensajes de aliento y su gran amor hacia mí.

Dedicatoria Leduan

Esta tesis se la dedico a la mujer más importante de mi vida, esa mujer que siempre ha estado presente cuando lo he necesitado, que siempre me ha apoyado incondicionalmente, que ha hecho todo lo posible para ayudarme a conseguir este sueño, ¡Gracias por todo Mamá!

A ese hombre que ha sido una guía para mí toda la vida, que siempre me ha aconsejado, que me ha dado su apoyo incondicional siempre, que cada vez que he necesitado algo ha estado presente, y que he compartido con él muchos de los momentos más felices de mi vida. ¡Gracias Papá por ser el mejor padre que se puede pedir!

Por último, a mi abuela Margot, que siempre me dio todo el cariño del mundo, y aunque ya no esté aquí conmigo, sé que desde allá arriba me cuida, ¡Gracias Abuelita, te quiero, y siempre te llevo en mi corazón!

Agradecimientos Arletis

Quiero agradecer en este momento especial a todas aquellas personas que me han ayudado en la realización de este trabajo.

- *A mi madre porque las palabras no son suficientes para expresar todo lo que siento por ella. Porque ha estado conmigo en cada momento importante y me ha enseñado que lo más importante para triunfar en la vida es perseverar.*
- *A mis abuelos, que gracias a sus enseñanzas, los mensajes de aliento y su gran amor hacia mí me han guiado siempre para afrontar la vida.*
- *A mi padrastro Victor porque quererme y cuidarme como a su hija y por apoyarme en cada decisión que tomo.*
- *A Carlos, porque sin su ayuda durante todos estos años no hubiera sido posible llegar hasta aquí.*
- *A una de mis mejores amigas, Mi Tía Griselis. A mi tío Rolando y mis dos queridos primos Roi y Gladi.*
- *A mis hermanitas del alma, por la fuerza y el deseo de superación que inspiran en mí.*
- *A mi primo Deivi por ser el hermano varón que nunca tuve.*
- *A Rosme porque me ha demostrado que los verdaderos amigos sí existen y estarán siempre presentes en tu vida.*
- *A Roi y Adri por quererme y cuidarme como una hermana menos. Las palabras no son suficientes para expresar todo el amor que siento por ustedes dos que llenan mi vida de alegría. A Roi que aunque a veces me vuelve loca no podría vivir sin él a mi lado.*
- *A Roly, Ruben, Rodni, Diomne, Arle, Yaimara, Claudia, Xiuny y Oscar por compartir también tan buenos momentos conmigo.*
- *A mi amigo y compañero de tesis Leduan por confiar en mí y brindarme todo su apoyo.*
- *A mis tutores Coca y Orlando por tanta dedicación y apoyo en este último año.*
- *A mi amigo del alma Jesús por ser tan especial conmigo y por todo su cariño y compañía.*

Agradecimientos Leduan

Quiero agradecer a mis padres por ser el maravilloso ejemplo que siempre han sido, a Jesús, que siempre me aconsejó mucho en mi vida y ha sido un gran ejemplo a seguir también, gracias por ser un padre más para mí. A mi abuela Leo, quien quiero con mi alma, que siempre me ha dado todo su amor, que se pasa la vida pensando en qué necesito para tratar de ayudarme, ¡Gracias Abue! A una de las personas más sabias que he conocido, alguien que me ha aconsejado mucho sobre las cosas de la vida y que siempre me ha dado su apoyo, ¡Gracias Abuelo! A mi hermanita que, aunque a cada rato discutimos, la quiero muchísimo, ¡Gracias mi flaca! A mi hermano, quien también me ha dado muchos consejos en la vida, y aunque ha estado lejos, siempre me ha apoyado y ha tratado de ayudarme en todo lo que puede, ¡Te quiero Hermano! A mi tía Koki y mi primo Leony que me han ayudado muchísimo cuando lo he necesitado ¡Gracias! A todos, los quiero y siempre los tendré en mi corazón.

Quiero agradecer también a quien, más que un amigo, ha sido un hermano para mí desde que lo conocí en primer año, alguien que me ha demostrado que puedo confiar en él con los ojos cerrados y que ha estado ahí para mí en las buenas y en las malas, gracias por todo Felo. Quiero agradecer a Lili, a Wendys y a Claudia, mis tres hermanitas de la universidad, por toda la ayuda que siempre me brindaron, todos los consejos que me dieron y los buenos momentos que pasamos, las quiero mucho chicas, nunca las olvidaré. A Ermis, quien ha sido un amigo incondicional también desde que nos conocemos, gracias por ser el amigo que siempre has sido. A Eliuvis, el primer ingeniero del piquete, otro buen amigo que conocí en esta universidad, y de los más jodedores que he conocido. A mi pequeña holguinera, que me demostró que no estaba equivocado al elegirla como compañera de tesis, pues no creo que pudiera haber encontrado alguien mejor, gracias por ser una excelente compañera y la gran amiga que siempre has sido, sabes que aquí me tienes para lo que sea. A mi tutor, Coca, que nos ayudó mucho y gracias a él todo salió como esperábamos. También a nuestro tutor Orlando que nos ayudó mucho cuando lo necesitábamos. A todas las amistades que conocí durante estos años, Erlis, Brian, Alik, al Lachy, Edward, al Erne, Víctor, al Yase y a Yaimara, a Papo, a Rodnier, a Diomne, a todos mis compañeros de aula y de apartamento que siempre me hacían reír entre el chucho y la jodedera, a todos, me alegro de haberlos conocido.

Resumen

Las nuevas tecnologías de la información y las comunicaciones han incidido cada vez más en el desarrollo de sistemas inteligentes utilizados en el proceso docente educativo. Sin embargo aún quedan lagunas en la utilización de los mismos en varias ramas de interés como la Inteligencia Artificial. La presente investigación se realiza a partir de la necesidad de que exista un sistema informático, basado en la programación lógica, que permita el estudio práctico del tema sistemas basados en el conocimiento en una de las asignaturas de la disciplina Inteligencia Artificial en la Universidad de las Ciencias Informáticas. De ahí que el objetivo del presente trabajo sea desarrollar un Agente Lógico para la resolución práctica de problemas en la disciplina Inteligencia Artificial de la Universidad de las Ciencias Informáticas. El desarrollo de la aplicación se basó en el uso de la metodología de desarrollo Programación Extrema y está compuesta por dos módulos: el primero basado en reglas y el otro basado en casos. Finalmente se llevó a cabo la validación de la propuesta a través de los métodos definidos en la investigación.

Palabras clave: agente lógico; inteligencia artificial; programación lógica; sistema inteligente.

Índice de contenidos

Índice de ilustraciones

Índice de tabla

Introducción

El vertiginoso desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC´s) ha cambiado nuestra forma de vida, impactando en muchas áreas del conocimiento tales como la salud, telecomunicaciones, finanzas, política, administración, empresas, educación, vida cotidiana y diversión (1). En el área educativa, las TIC´s han demostrado que pueden ser de gran apoyo tanto para los docentes, como para los estudiantes. La implementación de la tecnología en la educación puede verse sólo como una herramienta de apoyo, no viene a sustituir al maestro, sino pretende ayudarlo para que el estudiante tenga más elementos (visuales y auditivos) para enriquecer el proceso de enseñanza aprendizaje.

En la actualidad existen muchas ramas de la informática ampliamente investigadas, pero también hay otras en las que falta mucho por indagar, algunas de estas áreas son la programación lógica y la Inteligencia Artificial (IA), pues el objeto de hacer razonar a las máquinas ya lleva varios años estudiándose, muchos de estos estudios son orientados al conocimiento, tales como: almacenar conocimiento y obtener un razonamiento. En este campo están los Sistemas Inteligente (SI), los cuales a través de los años han adquirido cierta experiencia, ayudando a las personas a obtener una solución a partir de un conocimiento previo.

Una de las preocupaciones constantes de las universidades y centros de estudios superiores es estar a la vanguardia en los métodos de enseñanza y ofrecer oportunidades de prácticas innovadoras que apoyen la preparación de sus estudiantes para su futura vida laboral. El continuo desarrollo tecnológico que las organizaciones viven, particularmente en las áreas de almacenamiento de información, recuperación y comunicación, altera la manera de enseñar y, por supuesto, de aprender. Por eso, la enseñanza dentro de las universidades debe permanecer también en constante actualización y así estar acorde con lo que en la "vida real" se utiliza, ofreciéndoles la ventaja de estar mejor adaptados al medio laboral que utiliza estos sistemas inteligentes en sus actividades cotidianas. Además las universidades deben ofrecer a sus estudiantes las herramientas para que ellos puedan explorar profundamente el medio laboral en el que se desarrollarán, pero desde su posición en el seno estudiantil. Con estas herramientas, ellos pueden diferenciar, analizar, y crear su propio aprendizaje a través de una experiencia directa con el medio, aumentando su capacidad de respuesta y su habilidad para responder a las demandas tecnológicas del medio.

Son varias las herramientas diseñadas con fines lúdicos o prácticos que se insertan en los cursos de IA en el mundo. Sin embargo, muy pocas son creadas específicamente para la enseñanza.

En la Universidad de las Ciencias Informáticas (UCI) se imparten dos asignaturas lectivas de Inteligencia Artificial. En las mismas se utilizan muy pocas herramientas para la enseñanza de dicha materia y las que hay presentan varios inconvenientes, sobre todo debido a que dichas herramientas no cuentan con una interfaz visual acorde a los gustos o intereses de los estudiantes y que cumpla con las exigencias de la docencia, lo que atenta contra la motivación y el propio aprendizaje de los estudiantes. Por otra parte el lenguaje Prolog, basado en la lógica de primer orden, que es uno de los lenguajes de programación que se imparte en la carrera, cuesta mucho trabajo a los estudiantes asimilarlo, más tomando en cuenta que para practicarlo hay que utilizar una aplicación por consola. Además se enseña en el primer tema de la IA1 y muy pocas veces se retoma en otros temas de la disciplina, siendo creado precisamente para resolver problemas de IA.

Uno de los temas tratados en la asignatura IA2 y que no cuenta con una herramienta que posibilite abordar el contenido de un modo menos teórico y sí de una forma más práctica es la de los Sistemas basados en el conocimiento; debido a esto los estudiantes tienen que realizar los ejercicios en la libreta, lo cual hace el trabajo muy tedioso.

Teniendo en cuenta lo anteriormente expuesto se plantea el siguiente **problema científico**: ¿Cómo contribuir a la práctica del tema Sistema Basados en el Conocimiento a través de la programación lógica en la disciplina Inteligencia Artificial de la Universidad de las Ciencias Informáticas?

La investigación tiene como **objeto de estudio:** Los sistemas inteligentes, constituyendo el **campo de acción:** Los sistemas inteligentes para la resolución práctica de problemas de Inteligencia Artificial utilizando la programación lógica.

Para darle solución a este problema se propone el siguiente **objetivo general**: Desarrollar un Agente Lógico para la resolución práctica de problemas en la disciplina Inteligencia Artificial de la Universidad de las Ciencias Informáticas.

Para resolver el problema científico se plantearon las siguientes **tareas de investigación**:

1. Elaboración de la fundamentación teórica mediante el estudio de las bibliografías relativas a los sistemas inteligentes para conocer su estructura y componentes principales.
2. Elaboración de la fundamentación teórica mediante el estudio de la bibliografía relativa a la programación lógica como vía para la construcción de sistemas inteligentes.
3. Diseño de un sistema genérico en Prolog con interfaz visual en Java para la resolución de problemas mediante razonamiento basado en casos o en reglas.
4. Implementación de un sistema genérico en Prolog con interfaz visual en Java para la resolución de problemas mediante razonamiento basado en casos o en reglas.
5. Realización de pruebas funcionales al sistema.

Además para todo el proceso de investigación y elaboración de este trabajo se tomará en cuenta la utilización de varios **métodos científicos de investigación** como:

- **Histórico-lógico:** Método que permitirá conocer los antecedentes y las tendencias actuales referidas a sistemas expertos basados en conocimiento, además de conceptos, términos y vocabularios propios del campo que contribuyen en gran medida al entendimiento del trabajo.
- **Analítico-sintético:** Mediante este método se podrá estudiar y analizar una serie de documentos y teorías relacionados con los SI basados en conocimiento que utilicen la programación lógica y extraer los elementos más importantes de estos, los componentes por los que están conformados, su funcionamiento y la interacción entre ellos.
- **Modelación:** Utilizado para representar cómo ocurren los procesos que se desean automatizar, además para elaborar los diferentes modelos definidos en la metodología escogida y que sirven de guía durante el desarrollo de la solución.

Los **métodos empíricos** utilizados para obtener información sobre el objeto de estudio fueron:

- **Consulta bibliográfica**: Permite la elaboración del marco teórico de la investigación fundamentada por la información consultada.
- **Pruebas de validación:** Confirma la veracidad y utilidad de la propuesta elaborada.

El presente trabajo de diploma está estructurado de la siguiente forma: resumen, introducción, tres capítulos de contenido, conclusiones, recomendaciones, referencias bibliográficas, glosario de términos y anexos. A continuación se hace una breve descripción del contenido de cada uno de los capítulos:

Capítulo 1 "Fundamentación Teórica": se hace un análisis de los principales temas concernientes al objeto de estudio, se analizan los principales conceptos y términos asociados al problema en cuestión, así como otras soluciones existentes. Se refleja el estado del arte del tema de investigación.

Capítulo 2 "Solución Propuesta": se exponen las características de la propuesta de solución. Se realiza un levantamiento de requisitos, se obtienen las historias de usuarios, se realiza el plan de entregas y las tarjetas CRC. Además se exponen los elementos del diseño de la solución propuesta.

Capítulo 3 "Implementación y Pruebas": se realiza la implementación de las historias de usuarios, se chequea el plan de iteraciones y se crean las tareas de ingeniería para implementar exitosamente cada historia de usuario. Además, se demuestra el cumplimiento del objetivo del trabajo a través del análisis del resultado de las pruebas realizadas a la solución.

Capítulo #1. Fundamentación teórica

1.1 Introducción

Durante una investigación es necesario obtener conocimiento teórico que facilite la realización de las actividades y así poder llevar a la práctica lo aprendido. Para ello se hace necesario realizar una profunda búsqueda bibliográfica en el tema de los Sistemas Inteligentes (SI).

En este capítulo se abordan los principales elementos teóricos que conforman los SI basados en la programación lógica. Se exponen los componentes principales que están presentes en un SI y el uso en estos de los agentes inteligentes, los cuales serán tratados más adelante en el **epígrafe 1.3**. Se caracterizan las diferentes metodologías de desarrollo de software, el lenguaje de programación y el entorno de desarrollo a utilizar.

1.2 Sistema inteligente

Un sistema experto es un programa que se comporta como un experto para algún dominio de aplicación, normalmente reducido. Debe ser capaz de explicar las decisiones que ha ido tomando y el razonamiento subyacente.

"Un SI es un sistema informático que simula el proceso de aprendizaje, de memorización, de razonamiento, de comunicación y de acción como mismo lo realiza un experto humano en cualquier rama de la ciencia" (2). También se dice que un SI se basa en el conocimiento declarativo (hechos sobre objetos, situaciones) y el conocimiento de control (información sobre el seguimiento de una acción).

Para comprender mejor el funcionamiento de un SI es de vital importancia analizar la estructura de estos sistemas, principales características y el porqué de utilizarlos.
Un SI está conformado por (2) una Base de conocimientos (BC) la cual contiene conocimiento modelado extraído del diálogo con un experto. Una Base de hechos (Memoria de trabajo) que es la que contiene los hechos sobre un problema que se ha descubierto durante el análisis. Posee además un Motor de inferencia que es el que modela el proceso de razonamiento humano. Además de un módulo de justificación que se encarga de explicar el razonamiento utilizado por el sistema para llegar a una determinada conclusión. Por último posee una interfaz de usuario que se encarga de permitir la interacción entre el SI y el usuario y se realiza mediante el lenguaje natural.

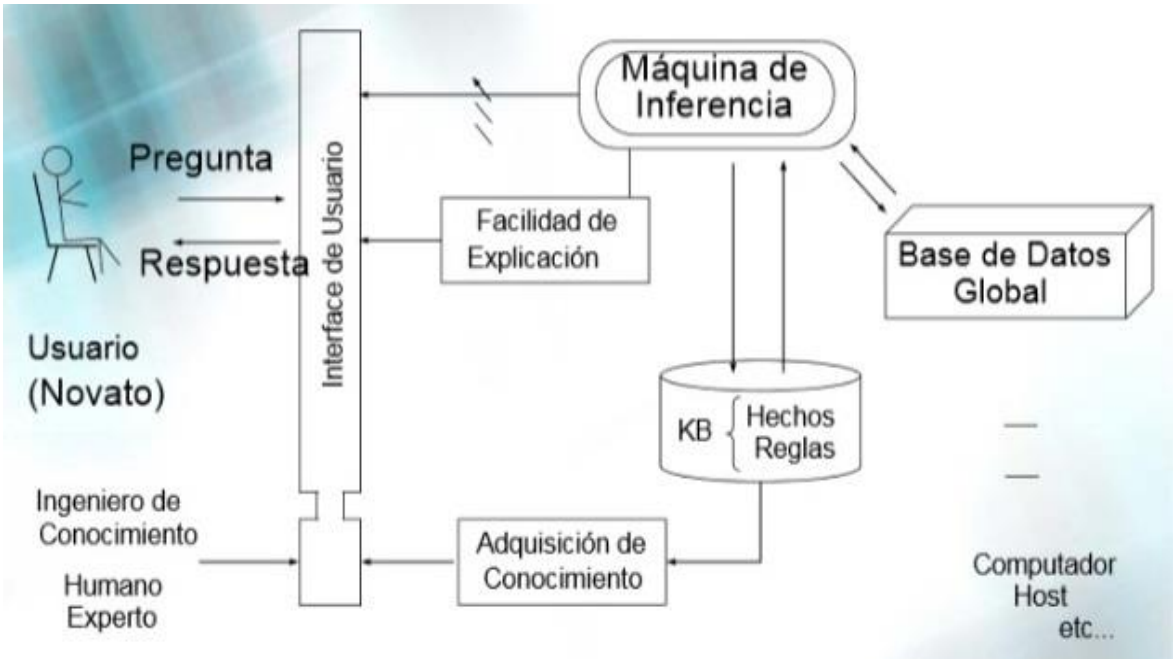


Ilustración 1: *Estructura de un SI (3)*

Con la ayuda de estos sistemas, personas con poca experiencia pueden resolver problemas que requieren un conocimiento especializado. Además, los SI pueden obtener conclusiones y resolver problemas de forma más rápida que los expertos humanos. Los mismos razonan, pero en base a un conocimiento adquirido y no tienen sitio para la subjetividad.

Los SI poseen las siguientes características, en menor o mayor grado (4):

- **Razonamiento guiado por las metas y encadenamiento hacia atrás.** Una técnica de inferencia que usa las reglas IF-THEN para descomponer las metas en submetas más fáciles de probar.
- **Manejo de incertidumbre.** La habilidad del SI para trabajar con reglas y datos que no son conocidos con precisión.
- **Razonamiento guiado por los datos y encadenamiento hacia adelante.** Una técnica de inferencia que usa las reglas IF-THEN para deducir soluciones a un problema a partir de los datos iniciales disponibles.
- **Representación de datos.** La forma en que los datos específicos a un problema dado son almacenados y accedidos por el SI.
- **Interfaz del usuario.** La parte del SI que se usa para una interacción más amigable con el usuario.

A continuación se muestran los dos tipos de SI fundamentales en los que se sustenta la investigación para una mejor comprensión de lo que se desea lograr:

- **Sistema Inteligente Basado en Reglas (SBR):**

Los SBR son las estructuras de sistemas expertos más comprensibles para la lógica humana, se definen a partir de un conjunto de objetos, que representan las variables del modelo, vinculadas mediante un conjunto de reglas encadenadas que representan las relaciones entre variables. Muchas de las situaciones con las que el ser humano se enfrenta a menudo, están gobernadas por reglas deterministas; en este ámbito los SBR constituyen una herramienta eficiente para la solución de estos problemas.
Las reglas describen lo que se debe hacer, desde el punto de vista de un experto en un ámbito concreto. Estas reglas pueden ser expresadas como reglas simples (de tipo **SI <<Condiciones>> ENTONCES <<Acciones>>**), como cuadro de decisión o como árbol de decisión.

En los SBR existen varios problemas los cuales caen dentro de una de las siguientes categorías: encadenamiento infinito, incorporación de conocimiento nuevo contradictorio y modificación de reglas existentes. Otros problemas pueden ser: ineficiencia, opacidad (dificultad de establecer relaciones), adaptación al dominio (rápido crecimiento del número de reglas) (5).

A pesar de las desventajas anotadas, los SBR han permanecido como los esquemas más comúnmente utilizados para la representación del conocimiento. Como ventajas significativas se pueden mencionar las siguientes: modularidad, uniformidad y naturalidad para expresar el conocimiento.

- **Sistema Inteligente Basado en Casos (SBCa):**

Consta de un motor de inferencia que modela el proceso del razonamiento humano y que –a diferencia de otros campos de la IA– son capaces de utilizar conocimiento específico de experiencias previas para poder resolver un problema actual.
Un SBCa se implementa mediante el siguiente ciclo:

- ✓ **Captura:** Del problema o caso actual con sus características.
- ✓ **Recupera:** El motor de inferencia realiza una búsqueda de casos similares y se encuentra el más parecido al problema a solucionar (‘vecino más cercano’).
- ✓ **Reutiliza:** Con base en el caso recuperado, se propone una solución al problema nuevo.
- ✓ **Revisa:** Se valida si la solución propuesta está considerada en la base de casos existentes.

- ✓ **Retiene:** Se aprende del problema actual para resolver problemas futuros.

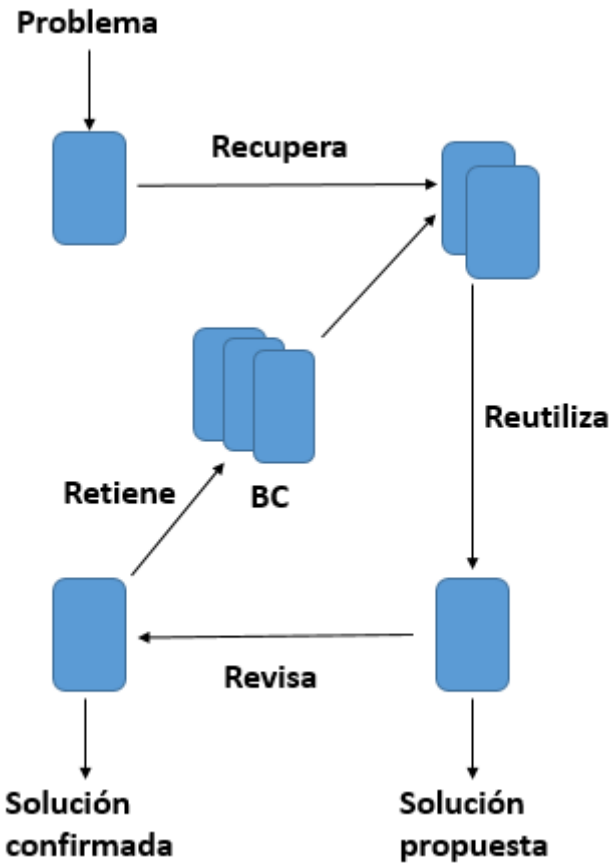


Ilustración 2. *Ciclo de un SBCa. (6)*

1.2.1 Ejemplos de SI

Para entender por qué y cuándo nacen los SI es necesario conocer un poco de historia sobre la IA, de lo cual ya se ha mencionado algo. Estos se desarrollaron al reconocerse la importancia primordial que tenían los conocimientos particulares de un dominio para que los métodos formales de búsqueda e inferencia sean eficaces en la solución de problemas. A continuación se muestran los SI más importantes en la historia de la IA y los que dieron paso a un desarrollo vertiginoso en esta rama.

- **Dendral**

Es un SI que permite la inferencia de estructuras moleculares a través de un proceso de búsqueda de generación y prueba jerárquica que se divide en tres partes funcionales: plan, generación y prueba. Su base de conocimientos se desglosa en dos conjuntos de reglas correspondientes a cada una de las fases de desarrollo del sistema (7) .

Fue el primer SI en ser utilizado para propósitos reales, al margen de la investigación computacional, y durante aproximadamente 10 años, el sistema tuvo cierto éxito entre

químicos y biólogos, ya que facilitaba enormemente la inferencia de estructuras moleculares, dominio en el que Dendral estaba especializado.

Su implementación no separaba de forma explícita el conocimiento del motor de inferencia. Sin embargo, pronto se convirtió en uno de los modelos a seguir por muchos de los programadores de sistemas expertos de la época.

- **Mycin**

Mycin es un SI de propósito específico, el cual fue uno de los primeros SI que se usaron para diagnosticar enfermedades en medicina. El sistema podía identificar bacterias que causaban severas infecciones, tales como la *bacteremia* y la *meningitis*. Igualmente, podía recomendar antibióticos dosificados, basándose en el peso del paciente. El nombre del programa derivó de los antibióticos que tienen muchas veces el sufijo “mycin”. El sistema también se usó para diagnosticar enfermedades infecciosas de la sangre (8). Además, era capaz de “razonar” el proceso seguido para llegar a estos diagnósticos y de recetar medicaciones personalizadas a cada paciente (según su estatura, peso, etc.).

- **Prospector**

Está diseñado como sistema de propósito específico para la consulta de los geólogos en las primeras etapas de investigación de un terreno en busca de yacimientos minerales. Los datos son principalmente observaciones geológicas superficiales y se supone que pueden ser inciertos e incompletos. El programa brinda a los usuarios posibles interpretaciones e identifica qué observaciones adicionales podrían ser útiles para alcanzar una conclusión más segura.
Prospector es un descendiente de Mycin que condujo a una nueva herramienta de construcción de sistemas. Además, va más lejos que Mycin en varios aspectos importantes.

Con el desarrollo de la Inteligencia Artificial se han ido desarrollando varios sistemas inteligentes de propósito general, siendo uno de los más reconocidos el Weka. Esta aplicación es una plataforma de software para el aprendizaje automático y la minería de datos escrito en Java. Weka es software libre distribuido bajo la licencia GNU-GPL. Una de las propiedades más interesantes de este software, es su facilidad para añadir extensiones, modificar métodos, etc (9).

Weka soporta varias tareas estándar de minería de datos, especialmente, preprocesamiento de datos, clustering, clasificación, regresión, visualización y selección. No puede realizar minería de datos multi-relacional, pero existen aplicaciones que pueden convertir una colección de tablas relacionadas de una base de datos en una única tabla que ya puede ser procesada con Weka.

Los principales resultados siguen siendo en sistemas de propósito específico, desarrollados para varias ramas de la economía, los servicios, la salud y la educación. La programación lógica ha jugado un importante papel en este sentido. Los siguientes son algunos de los ejemplos más recientes de investigaciones relacionadas con la programación lógica y los sistemas inteligentes (10), (11), (12), (13), (14), (15), (16), (17), (18) y (19).

En Cuba se han desarrollado varios sistemas inteligentes siendo muy referenciado el Sistema Inteligente de Selección de Información (SISI). Sistema cubano de propósito general que está destinado para estudiantes de medicina, enfermería y estomatología, residentes y médicos. Permite la realización de consultas y diagnósticos médicos con la combinación de las técnicas proporcionadas por los sistemas basados en casos y las redes neuronales. Una ventaja muy importante de la misma tiene que ver con su utilidad para la consulta de información, pero no está concebido para la identificación de riesgos de ningún tipo (20).

En (21) se exponen un conjunto de Reglas que permiten explotar el conocimiento de los esquemas de relés para la localización de la sección en fallo. Este realiza el mecanismo de inferencia mediante lenguaje Prolog y estima la sección en fallo con la justificación de cada operación de los relés e interruptores. Para la identificación de las secciones en fallo se prueban un conjunto de hipótesis las cuales están establecidas en forma jerárquica, de forma tal que puedan dar respuesta a cualquier situación creada, aun en caso de fallos múltiples o fallos de operación de los relés e interruptores.

El Sistema de diagnóstico para la estimación de secciones en fallo en sistemas eléctricos de potencia está escrito en Prolog. La tarea básica de este sistema es entrenar a los despachadores y utilizarlo como una herramienta de apoyo a la toma de decisiones del despachador ante situaciones de emergencias complejas con un ambiente computacional de usuario amigable (21).

El Sistema para la Enseñanza de Sistemas Expertos (SESE) fue desarrollado en la Universidad Central de Las Villas (Santa Clara, Cuba) como respuesta a la necesidad de brindar a los aprendices un ambiente interactivo que permita vivir experiencias directas y relevantes al aprendizaje de conceptos y habilidades de programación de sistemas expertos (22). También el sistema LogSim de apoyo al aprendizaje de la programación lógica, el cual se viene utilizando en la carrera de Licenciatura en Ciencias de la Computación de dicha universidad. Mediante la simulación del comportamiento del lenguaje Prolog se propicia que los alumnos lleguen a dominar sus mecanismos más importantes (23).

También se han llevado a cabo investigaciones teóricas en nuestro país en cuanto a sistemas inteligentes. Recientemente fue publicada una arquitectura para desarrollar de forma unificada tanto el razonamiento basado en reglas como el basado en casos considerando rasgos borrosos. El modelo permite usar varios de los procedimientos desarrollados para el razonamiento borroso sustentado en reglas para tal razonamiento (24).

El análisis de los sistemas anteriores muestra la importancia y utilidad de los sistemas expertos, además de la vigencia en las investigaciones actuales. Sin embargo para la enseñanza de este tema en la asignatura IA se requiere un sistema diseñado en Prolog, con interfaz visual agradable y de código abierto, además que sea de propósito general y permita la modificación e inclusión de nuevos métodos y algoritmos. Debido al creciente desarrollo y la gran demanda que tienen estos sistemas en el mercado actual, estos dieron paso a que surgiera un nuevo concepto: Agente Inteligente, estos sistemas no se consideran expertos en una sola rama, sino que serán capaces de actuar y razonar de forma autónoma según el conocimiento previo y los cambios en su entorno.

1.3 Agente inteligente

Se define como agente inteligente a aquella entidad capaz de captar la información de su entorno, procesar la misma y posteriormente actuar en correspondencia a dicho procesamiento, todo esto con la ayuda de sensores y actuadores (elementos que reaccionan a un estímulo realizando una acción).

Las cualidades que tendría un agente inteligente ideal, según James A. Hendler (25), serían:

- **Comunicativo:** el agente debe entender las necesidades, objetivos y preferencias del usuario para que éste pueda realizar su función correctamente.
- **Capaz**: El agente no sólo debe proporcionar una información, sino también un servicio, es decir, debe tener capacidad para hacer cosas.
- **Autónomo**: el agente, además de comunicarse, debe poder interactuar con el entorno, tomando decisiones y actuando por sí solo, limitando sus acciones según el nivel de autonomía permitida por el usuario.
- **Adaptativo**: Debe ser capaz de aprender del entorno: preferencias de usuarios, fuentes de información y de otros agentes.

Es necesario, para una mejor comprensión de lo que es un agente inteligente conocer de sus principales características.

Los agentes inteligentes se han definido de diferentes maneras y poseen las siguientes características (4):

- Aprender nuevos problemas e incrementar normas de solución.
- Capacidad de adaptación en línea y en tiempo real.
- Ser capaz de analizar condiciones en términos de comportamiento, el error y el éxito.
- Aprender y mejorar a través de la interacción con el medio ambiente (realización).
- Aprender rápidamente de grandes cantidades de datos.
- Deben estar basados en memoria de almacenamiento masivo y la recuperación de dicha capacidad.

Un agente inteligente debe ser capaz de aprender a través de sus experiencias obtenidas mediante la interacción con el entorno en que se encuentra.

1.4 Aprendizaje

El aprendizaje es una de las funciones mentales más importantes en humanos, animales y sistemas artificiales (26). Este proceso permite adquirir o modificar habilidades, destrezas, conocimientos, conductas o valores como resultado del estudio, la experiencia, la instrucción, el razonamiento y la observación. Es todo aquel

conocimiento que se va adquiriendo a través de las experiencias de la vida cotidiana, en la cual el usuario se apropia de los conocimientos que cree convenientes para su aprendizaje.

El aprendizaje ayuda a los expertos a mejorar su calidad profesional al aprender de las nuevas experiencias aumentando su eficacia para los futuros problemas. Esto es altamente deseable también en los SI por lo que este es el programa encargado de analizar las razones de sus éxitos y fracasos para perfeccionar la base de conocimientos y ser más eficaz y efectivo en el futuro, como así también incluir nuevos conocimientos. Esto trae consigo que se cree la incertidumbre debido a que el grado de certeza de una hipótesis puede ser o no cierta.

Incertidumbre

Todo sistema que tenga por objetivo simular las interacciones entre componentes inteligentes y autónomos debe considerar el manejo de incertidumbre e inconsistencia. La incertidumbre se puede resumir como la información incompleta, fuentes poco confiables, detalles y hechos importantes que cambian, hechos imprecisos, vagos o difusos. La gente llega a soluciones razonables a pesar de todo.

Desde los inicios se eligió la probabilidad como medida para tratar la incertidumbre, pero se encontraron problemas, debido fundamentalmente al uso simplificado de las hipótesis de independencia, utilizadas para reducir la complejidad. Consecuentemente, surgieron medidas alternativas a la probabilidad como los factores de certeza, credibilidades, plausibilidades, necesidades o posibilidades. Sin embargo la probabilidad sigue siendo la forma más aceptada, sobre todo por ser la menos dependiente del factor humano para tratar la incertidumbre; además permite a los agentes calcular por sí mismo valores de certeza probabilística, incluso los representados mediante lógica de primer orden.

1.5 Lógica de primer orden (LPO)

“La LPO, también llamada lógica de predicados o cálculo de predicados, es un sistema formal diseñado para estudiar la inferencia en los lenguajes de primer orden” (27).

El lenguaje de la LPO está construido sobre objetos y relaciones. Precisamente por este motivo ha sido tan importante para las Matemáticas, la Filosofía y la Inteligencia Artificial porque se puede pensar en ello de forma utilitaria como en el tratamiento con objetos y de las relaciones entre estos. La LPO también puede expresar hechos acerca de algunos o todos los objetos de un universo de discurso (28).

La principal diferencia entre la lógica proposicional y la de primer orden se apoya en el compromiso ontológico realizado por cada lenguaje (es decir, lo que asume cada uno acerca de la naturaleza de la realidad). Por ejemplo, la lógica proposicional asume que hay hechos que suceden o no suceden en el mundo. Cada hecho puede estar en uno de los dos estados: verdadero o falso. La LPO asume mucho más, a saber, que el mundo se compone de objetos con ciertas relaciones entre ellos que suceden o no suceden.

La LPO tiene el poder de expresar de forma sencilla las relaciones entre objetos, esto le brinda a los SI una mayor potencialidad a la hora de realizar inferencias, ya sea por SBCa o SBR.

La LPO se considera la base fundamental de la Programación Lógica (PL), término que se aborda en el siguiente epígrafe.

1.6 Programación Lógica

La Programación Lógica (PL) estudia el uso de la lógica para el planteamiento de problemas y el control sobre las reglas de inferencia para alcanzar la solución automática.

La PL, junto con la funcional, forma parte de lo que se conoce como Programación Declarativa, es decir la programación consiste en indicar cómo resolver un problema mediante sentencias, en la PL, se trabaja en una forma descriptiva, estableciendo relaciones entre entidades, indicando no cómo, sino qué hacer, entonces se dice que la idea esencial de la PL es:

Programa= Lógica + control

Lógica (programador): hechos y reglas para representar conocimiento.

Control (interprete): deducción lógica para dar respuestas a soluciones. También conocida como máquina de inferencia.

La PL intenta resolver lo siguiente:

Dado un problema S, saber si la afirmación A es solución o no del problema o en qué casos lo es. Esta construye su base de conocimiento mediante reglas y hechos.

- **Regla:** implicación o inferencia lógica que deduce nuevo conocimiento, la regla permite definir nuevas relaciones a partir de otras ya existentes.

Ej.:

Mortal (x): - humano(x).

X es mortal si x es humano.

- **Hecho:** declaración, cláusula o proposición cierta o falsa, el hecho establece una relación entre objetos y es la forma más sencilla de sentencia.

Ej.:

Humano (Sócrates); Sócrates es humano.

Ama (Juan, María); ama Juan a María.

- **Consulta:** se especifica el problema, la proposición a demostrar o el objetivo partiendo de que los humanos son mortales y de que Sócrates es humano, deducimos que:

Sócrates es mortal

Mortal (x): - humano(x);- los humanos son mortales; regla

Humano (Sócrates); Sócrates es humanos; hecho

Sócrates es mortal; consulta

Algunas de las desventajas que proporciona la PL son:

- Poco eficientes.
- Poco utilizado en aplicaciones reales.

A pesar de sus desventajas, la PL también presenta importantes ventajas como:

- Simplicidad.
- Cercanía a las especificaciones del problema realizada con lenguajes formales.
- Sencillez, potencia y elegancia.
- Metodología rigurosa de especificación.
- Sencillez en la implementación de estructuras complejas.

1.7 Metodología para el desarrollo de software

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software (29). El objetivo de una metodología de desarrollo de software es garantizar la eficacia y la eficiencia en el proceso de generación de software.

1.7.1 Proceso Unificado de Desarrollo (RUP)

El proceso unificado conocido como RUP, es un modelo que permite el desarrollo de software a gran escala, mediante un proceso continuo de pruebas y retroalimentación, garantizando el cumplimiento de ciertos estándares de calidad. Aunque con el inconveniente de generar mayor complejidad en los controles de administración del mismo. Sin embargo, los beneficios obtenidos recompensan el esfuerzo invertido en este aspecto.

El proceso de desarrollo constituye un marco metodológico que define en términos de metas estratégicas, objetivos, actividades y artefactos (documentación) requerido en cada fase de desarrollo. Esto permite enfocar esfuerzo de los recursos humanos en términos de habilidades, competencias y capacidades a asumir roles específicos con responsabilidades bien definidas.

RUP se divide en cuatro fases (30):

- Inicio (Define el alcance del proyecto).
- Elaboración (definición, análisis, diseño).
- Construcción (implementación).
- Transición (fin del proyecto y puesta en producción).

Entre las principales características que presenta RUP se encuentra la forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo). Pretende implementar las mejores prácticas en Ingeniería de Software. Presenta un desarrollo iterativo y facilita la administración de requisitos. Usa arquitecturas basadas en componentes y verificación de la calidad del software (30)**.**

Metodología de desarrollo de software que está basada en componentes e interfaces bien definidas, y junto con el Lenguaje Unificado de Modelado (UML), constituye la

metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

1.7.2 Metodologías para el desarrollo de Agentes.

Existen actualmente un gran número de metodologías propuestas para desarrollar sistemas multi-agente (SMA): TROPOS (31), Ingenias (32), entre otras (33). Esta diversidad dificulta en cierto grado la elección de una metodología para desarrollar sistemas agentes según las características que se esperan del mismo. Ingenias es una metodología de propósito general y aplicable a desarrollos industriales, detalla con profundidad las etapas de análisis, diseño e implementación (33), (32). A pesar de las facilidades que brinda el modelado con Ingenias para obtener un SMA, el utilizar RUP como punto de partida, en la captura de requisitos, dificulta tener un hilo conductor desde el inicio del proyecto hasta el final, ya que RUP está enfocada a la orientación a objetos. Además, la herramienta de desarrollo de Ingenias no soporta esta fase (34).

Esta metodología es una evolución de las ideas de la Metodología de Sistemas de Ingeniería de Software de agentes (MESSAGE[1], *por sus siglas en inglés*) (35) . INGENIAS profundiza en los elementos mostrados en el método de especificación, en el proceso de desarrollo, además de incorporar nuevas herramientas de soporte y ejemplos de desarrollo. Define un conjunto de meta-modelos que ayudan a describir el sistema.

Esta metodología es bastante reciente. Dispone de varios desarrollos donde se experimenta con ella, queda por demostrar su viabilidad real. Sugiere añadir capacidades de descripción de ontologías pero esta metodología no contempla el desarrollo de ontologías. Mediante la herramienta de soporte Ingenias IDE, se facilita su uso pero se sigue requiriendo que el desarrollador revise la documentación para entender qué hace cada entidad y cuál es el propósito de cada relación (35).

Ingenia tiene como principios (36):

1. Agentes como paradigma de modelado
   - Conceptos de más alto nivel en objetos y más cercanos al dominio.
   - Se pueden considerar adaptaciones específicas a dominios de aplicación particulares.

[1] Adopta el Proceso Unificado de Desarrollo de Software.

- Los aspectos organizativos e intencionales reducen el salto de especificación de requisitos a implementación.

2. Implementación sobre distintos tipos de plataforma

- Un modelo de SMA se puede implementar sobre una plataforma de agentes o sobre un entorno de objetos tradicional.

- La metodología facilita y promueve el desarrollo de herramientas de generación de código que faciliten el paso del modelo (análisis y diseño) a la implementación.

3. Contempla la evolución de la tecnología de agentes

- Adaptabilidad a nuevos lenguajes y estándares (p.ej. AUML).

Una de las ventajas de INGENIAS es que se integra fácilmente con RUP. En la **Tabla 1** podemos ver las asociaciones entre los elementos del RUP.

Tabla 1: *Asociación entre INGENIAS y RUP (37).*

Entidad SMA	Entidad RUP
Agente	Clase
Organización	Arquitectura
Grupo	Subsistema
Interacción	Escenario
Roles, tareas y flujos de trabajo	Funcionalidad

Según el estudio realizado, Ingenias plantea que la captura de requisitos debe hacerse como se realiza en RUP. En el caso de lo que se quiere realizar en la presente investigación esta no es la forma adecuada para capturar los requisitos, teniendo en cuenta que se quiere construir un Agente y que RUP es un proceso dirigido fundamentalmente al paradigma de orientación a objetos.

Las tareas del proceso de captura de requisitos están descritas en lenguaje natural y diagramas de actividad. Como ocurre con otras propuestas, Ingenias no brinda guías precisas para identificar sus entidades de modelado con conceptos del mundo real.

Debido a que Ingenias es una metodología para SMA no se considera factible para el desarrollo del sistema ya que este está basado en un único agente el cual no requiere de todo el análisis que conlleva dicha metodología, como por ejemplo la relación existente entre agentes.

1.7.3 Extreme Programming (XP)

XP es una metodología de desarrollo de la ingeniería de software formulada por Kent Beck en 1999. Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, XP se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los valores originales de la programación extrema son: simplicidad, comunicación, retroalimentación (*feedback*) y coraje. Un quinto valor, respeto, fue añadido en la segunda edición de *Extreme Programming Explained* (12).

Las características fundamentales de XP son (12):

- **Desarrollo iterativo e incremental:** pequeñas mejoras, unas tras otras.
- **Pruebas unitarias continuas:** frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- **Programación en parejas:** se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. La mayor calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata.
- **Frecuente integración del equipo de programación con el cliente o usuario:** Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- **Corrección de todos los errores:** antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
- **Refactorización del código:** reescribir ciertas partes de código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.

Diferencias entre las metodologías XP y RUP (38):

En la metodología XP no se definen todos los requerimientos al principio del ciclo de vida, se pueden aplicar cambios de requerimientos durante el ciclo de vida del software. XP propone el trabajo en equipo, es decir la programación en pareja y revisando el código, esto brinda la posibilidad de aportar nuevas ideas a la forma de programar e ir corrigiendo el código al mismo tiempo que se programa. Además, XP propone que un representante del cliente trabaje junto con el equipo.

Ilustración 3.*Comparación entre RUP y XP (38).*

	XP	RUP
Selección de la metodología.	Requisitos cambiantes.	Comunicación entre equipos.
	Proyecto con alto grado de riesgo.	Complejidad de desarrollo de acuerdo al tamaño del proyecto.
	Grupos pequeños de programadores.	Configuración y control de cambios.

1.7.4 Selección de la metodología de desarrollo de software

✓ La metodología elegida para el análisis y diseño del sistema basado en el conocimiento es XP porque es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo del software. Además para su selección se tuvieron en cuenta las siguientes características:

- XP ofrece una solución factible para proyectos con requisitos muy cambiantes, propone la aplicación disciplinada de las diferentes prácticas y el uso de tecnologías con un ciclo rápido de realimentación y que soporten fácilmente el cambio.

- La aplicación disciplinada de las diferentes prácticas, por ejemplo la programación en pareja (2 integrantes) donde especifica que toda la producción de código debe realizarse en parejas de programadores, lo cual tiene como ventajas que muchos errores sean detectados conforme son introducidos en el código, por consiguiente la tasa de errores del producto final es más baja, los diseños son mejores y el tamaño del código menor. Otra

práctica utilizada es la definición de estándares de programación con el objetivo de mantener el código legible para los miembros del equipo, facilitando los cambios. Además la propiedad y responsabilidad colectiva porque cualquiera puede cambiar cualquier código en cualquier momento tomando la responsabilidad por el sistema completo y se integra a intervalos cortos de tiempo, para que los conflictos aparezcan lo antes posible.

- Es una metodología sencilla tanto en su aprendizaje como en su aplicación, permitiendo con esto al equipo de desarrollo minimizar el tiempo empleado para su familiarización, entendimiento y puesta en práctica de la misma.

1.8 Tecnologías y herramientas para dar solución al problema planteado

Uno de los aspectos más importantes a tener en cuenta en el proceso del desarrollo de software es la selección de las herramientas y tecnologías que se adecuen al sistema que se quiera realizar. Las tecnologías y herramientas que existen en la actualidad han venido a revolucionar la forma de automatizar los aspectos claves en el desarrollo de sistemas informáticos, proporcionando toda una gama de componentes necesarios para el desarrollo de las aplicaciones, estas ofrecen además una gran plataforma de seguridad a los sistemas que las usan y han sido creadas con una gran exactitud en torno a las necesidades de los desarrolladores de sistemas para la automatización de procesos.

1.9 Lenguajes de programación para la interfaz visual

Java 8.0 es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases. Tiene como objetivo permitir que los desarrolladores puedan compilar una sola vez el programa y ejecutarlo en cualquier dispositivo, lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra (39). Se selecciona Java como lenguaje de programación a utilizar en el desarrollo de la solución.

1.10 Lenguajes de programación para el razonamiento lógico del sistema

Prolog es, de lejos, el lenguaje de programación lógica más extensamente utilizado. Se utilizó inicialmente como un lenguaje de prototipado rápido y para las tareas de manipulación de símbolos, como la escritura de compiladores y en el análisis sintáctico del lenguaje natural. Muchos sistemas expertos se han escrito en Prolog, para dominios legales, médicos, financieros y muchos otros (40).

Prolog es un lenguaje utilizado para implementar inteligencia artificial y sistemas expertos. Gran parte de su éxito se debe a su conveniencia por ser código abierto (modificable) y se obtiene fácilmente en Internet, además de su capacidad de deducción de respuestas para las consultas realizadas. La programación lógica es un paradigma de los lenguajes de programación en el cual los programas se consideran como una serie de aserciones lógicas (39).

Los programas en Prolog son conjuntos de cláusulas positivas escritas en una notación algo diferente a la estándar de la lógica de primer orden. Las cláusulas están escritas con la cabeza precediendo al cuerpo; «:-» se utiliza para las implicaciones a la izquierda, las comas separan los literales en el cuerpo, y el punto indica el final de la sentencia:

hermano(X,Y): -progenitor(M,Y), progenitor(M,X), macho(X).

La ejecución de los programas en Prolog se hace mediante el encadenamiento hacia atrás en primero en profundidad, donde las cláusulas se van intentando en el orden en el que se han escrito en la base de conocimiento. (40)

Escribir un programa en Prolog consiste en declarar el conocimiento disponible acerca de objetos, además de sus relaciones y sus reglas, en lugar de correr un programa para obtener una solución, se hace una pregunta, el programa revisa la base de datos para encontrar la solución a la pregunta, si existe más de una solución, Prolog hace un barrido para encontrar soluciones distintas. El propio sistema es el que deduce las respuestas a las preguntas que se le plantean, dichas respuestas las deduce del conocimiento obtenido por el conjunto de reglas dadas.

Características fundamentales del lenguaje Prolog (41):

- Basado en lógica y programación declarativa.
- No se especifica cómo debe hacerse, sino qué debe lograrse.
- Una característica importante en Prolog y que lo diferencia de otros lenguajes de programación, es que una variable sólo puede tener un valor mientras se cumple el objetivo.
- El programador se concentra más en el conocimiento que en los algoritmos.
- En Prolog, se llega a una solución infiriéndola desde algo ya conocido.
- Hay un conjunto de funciones incorporadas para la aritmética. Los literales que utilizan estos símbolos de función se «demuestran» mediante la ejecución de código, y no realizando inferencias adicionales. Por ejemplo, el objetivo «X es 4+3» tiene éxito con la X ligada a 7. Por el otro lado, el objetivo «5 es X+Y»

falla, ya que las funciones incorporadas no pueden resolver ecuaciones arbitrarias.

- Hay predicados incorporados que tienen efectos colaterales cuando se ejecutan. Estos son predicados de entrada y salida y predicados de acertar/retractar para modificar la base de conocimiento. Este tipo de predicados no tienen su homólogo en la lógica y pueden producir algunos efectos confusos, por ejemplo, si los hechos se aciertan en una rama del árbol de demostración que eventualmente falla.
- El Prolog permite un tipo de negación denominada "negación por fallo". Un objetivo negado "no P" se considera demostrado si el sistema falla al probar P. Así, la sentencia "vivo(X):- no muerto(X)" se puede leer como «Todo el mundo está vivo si no es probable que esté muerto».

Las ventajas de la forma declarativa de este lenguaje son claras (es más fácil pensar las soluciones y muchos detalles procedurales son resueltos automáticamente por el propio lenguaje) y podemos aprovecharlas (41). Una ventaja desde el punto de vista del usuario es la facilidad para programar ya que se pueden escribir programas rápidamente, con pocos errores originando programas claramente legibles, aun si no se conoce muy bien el lenguaje. Además, no hay que pensar demasiado en la solución del problema, ya que Prolog infiere sus respuestas basándose en las reglas declaradas dentro del programa. La modularidad porque cada predicado (procedimiento) puede ser ejecutado, validado y examinado independiente e individualmente. Prolog no tiene variables globales, ni asignación. Cada relación está auto contenida, lo que permite una mayor modularidad, portabilidad y reusabilidad de relaciones entre programas. El Polimorfismo que se trata de un lenguaje de programación sin tipos, con un alto nivel de abstracción e independencia de los datos (objetos). También, en Prolog se puede representar incluso los mismos programas Prolog como estructuras, utiliza un mecanismo de búsqueda independiente de la base de hechos. Aunque pueda parecer algo retorcido, es una buena estrategia puesto que garantiza el proceso de todas las posibilidades. Es útil para el programador conocer dicho mecanismo a la hora de depurar y optimizar los programas.

1.11 Plataforma de Desarrollo

Java FX 2.2 es una plataforma de desarrollo pensada para crear y desplegar aplicaciones para internet que funcionan en una gran cantidad de dispositivos (escritorio, navegadores y móviles).

Esta plataforma provee al programador grandes ventajas debido a que posee un motor de *renderización* web y un motor de multimedia de alto rendimiento. Además provee más de 50 componentes de diseño, formularios fácilmente personalizables mediante el uso de CSS. Lo que la diferencia de java es que esta plataforma le brinda estilo, diseño y personalidad a las aplicaciones.

1.12 Entorno de Desarrollo Integrado (IDE[2])

Netbeans 8.0 es un entorno de desarrollo integrado (IDE) gratuito y de código abierto multiplataforma con el que podremos programar en un buen número de lenguajes, desde Java o Ruby hasta C.

La posibilidad de visualizar tanto código como diseño convierte a **Netbeans** en una aplicación realmente versátil, llegando a ser útil incluso para la programación de aplicaciones web. (39)

SWI-Prolog 6.2.6 es una implementación en código abierto del lenguaje de programación Prolog. Posee un rico conjunto de características, bibliotecas (incluyendo su propia biblioteca para GUI), herramientas (incluyendo un IDE) y una documentación extensiva. El mismo funciona en las plataformas Unix, Windows y Macintosh (42).

1.13 Consideraciones finales del capítulo

En este capítulo se mostró una visión teórica de los principales conceptos que van a estar presentes durante todo el ciclo del trabajo de diploma, se realizó un estudio detallado sobre los SI, así como las tecnologías y herramientas a utilizar en la misma para dar cumplimiento a los objetivos propuestos y así poder crear un producto robusto y de calidad para el cliente, teniendo en cuenta las políticas de desarrollo de software presentes en la Universidad de las Ciencias Informáticas. Luego de realizado el estudio del arte se seleccionó como lenguaje de modelado el UML 2.0. Como lenguaje de programación se escogió Java 8 para el aspecto visual y Prolog para el procesamiento. En el caso del IDE de desarrollo se utilizará Netbeans 8.0 para la implementación de JavaFX y el SWI-Prolog para Prolog. Para el desarrollo del software se seleccionó como metodología XP.

[2] Por sus siglas en inglés (Integrated Development Environment)

Capítulo #2: Solución Propuesta

2.1 Introducción

En el presente capítulo se realiza una descripción del alcance que tendrá la propuesta de solución permitiendo tener una concepción general del sistema. En el mismo, se define y modelan los procesos del negocio que interviene en el SI, se expone el modelo conceptual donde se representan los conceptos fundamentales del dominio del problema y se identifican y definen los requisitos funcionales y no funcionales de la aplicación. Además, se especifican las condiciones y pasos a tener en cuenta para el correcto funcionamiento de la solución.

2.2 Descripción de la solución

La presente investigación pretende obtener como resultado el SI **Agente Lógico**, este sistema debe ser capaz de resolver problemas basados en reglas o basados en casos, dependiendo de los elementos que el usuario le brinde. Para ello debe realizar al menos parte de la inferencia siguiendo el paradigma de la programación lógica. A su vez debe permitir que el usuario diseñe el problema que pretende resolver y luego permitirle elegir entre el módulo basado en casos y el módulo basado en reglas para llegar a una solución.

Además se desea que el sistema cuente con inteligencia propia, por lo que este será desarrollado siguiendo un modelo de agente inteligente que permitirá el razonamiento del sistema. Este agente se encargará de todo el procesamiento lógico que llevará a cabo el sistema para llegar a una solución determinada. También le brindará algunas funcionalidades que ofrecerán comodidades al usuario, por ejemplo, saber cuáles son las Bases de Conocimiento (BC) más utilizadas y mostrárselas al usuario de primero, ir aprendiendo más sobre la BC en cuestión mientras más se use, permitiéndole dar respuestas más certeras y rápidas, entre otras características. El agente debe ser capaz de calcular el peso de cada uno de los rasgos en los casos almacenados, así como calcular un grado de certeza sobre los hechos almacenados y los inferidos.

Las variables almacenadas en la BC tendrán el siguiente formato:

variable(datos(nombre, tipo, [dominio], peso)).

En el caso de las variables nominales, el dominio estará compuesto por cada uno de los posibles valores que puede tomar la variable. En el caso de las variables de tipo real estará compuesto por el mínimo y el máximo valor que puede tomar la misma.

Además el usuario debe indicar cuál será la variable dependiente en la BC, ya que en la mayoría de los problemas que se resuelven de este tipo hay una variable cuyo valor depende del resto.

El formato que tendrá cada una de las reglas en el módulo SBR es el siguiente:

si **Condición** entonces **Conclusión con_certeza valor**

La condición estará compuesta por hechos interconectados entre sí mediante conectores lógicos, los cuales son “y”, “o”, “no”. Se tendrá un grado de certeza por cada hecho conocido y cada regla almacenada en la BC, dicha certeza estará comprendida en el rango [-1; 1]. Para determinar la certeza de la conclusión se utiliza la técnica Mínimo-Máximo para los conectores y el CTR-Mycin para determinar la contribución de la condición a la conclusión.

El módulo basado en casos será capaz de mostrar una explicación del razonamiento llevado a cabo para llegar a la solución mostrada, permitiéndole al usuario saber todo el análisis realizado por el agente para llegar a esa conclusión. El formato de dichos casos es el siguiente:

caso(datos(id, rasgo(valor1,certeza1),…, rasgo(valorN, certezaN))).

Nota: Los datos deben estar en el mismo orden en que fueron declaradas las variables en la BC.

Los valores de certeza indicados en la estructura anterior se calculan combinando una probabilidad a priori y una probabilidad condicional ingenua, la cual es calculada de la siguiente forma:

((Apariciones de ValorDecisión / Total de casos) + (Apariciones de ValorRasgo / Total de casos)) - ((Apariciones de ValorDecisión / Total de casos) * (Apariciones de ValorRasgo / Total de casos))

Donde:

- Apariciones de ValorDecisión es la cantidad de veces que aparece el valor del rasgo actual asociado a ese valor de decisión.
- Apariciones de ValorRasgo es la cantidad de veces que aparece el valor actual del rasgo en cada una de las instancias almacenadas.

Este módulo se encargará de clasificar un nuevo caso, ayudándose para esto de los casos almacenados en la BC. Se realiza una comparación con cada uno de los casos almacenados, para esto se utiliza el cuantificador universal *findall* el cual permite recuperar todos los casos almacenados, y mediante el predicado *univ* se transformó la estructura de datos de los casos a una lista, con la cual se calcula la semejanza de estos casos con el caso nuevo. Dicha semejanza se calcula mediante la siguiente fórmula:

$$S(C_n,C_i) = \sum_{j=0}^{N} w_j * \delta(O_{nj}, O_{ij}) \ / \ N$$

Donde:

- $S(C_n,C_i)$ es la semejanza entre el caso nuevo y el caso i de la BC.
- N es la cantidad de rasgos que posee cada caso.
- $\delta(O_{nj}, O_{ij})$ es la comparación entre el rasgo j del caso nuevo y el rasgo j del caso i en la BC.
- w_j es el peso del rasgo j.

Este peso se calcula mediante la ganancia de información, o sea, se calcula qué tanto aporta el rasgo j a la variable dependiente.

Estos valores de semejanza calculados tendrán asociados además un valor de certeza, el cual es calculado aplicando un Mínimo a las certezas de cada uno de los rasgos del caso que se está tratando y del caso a clasificar.

Luego, de estos casos se seleccionan los más semejantes mediante el umbral definido por el usuario. Esta lista de casos similares es tratada con Java, que es donde se lleva a cabo la adaptación luego de que Prolog se haya encargado de realizar todo el proceso de recuperación. Para este proceso de adaptación se realiza lo siguiente: Si el rasgo que se está clasificando es nominal su clasificación se hace determinando la moda en los casos recuperados y se aplica un Máximo entre las certezas de los casos que tengan como valor, en dicho rasgo, la moda, para de esta forma brindar un grado de certeza de la clasificación dada. Por otra parte, si el rasgo a clasificar es real, se determina su clasificación hallando el promedio, luego se aplica un Mínimo a las certezas de semejanza con el mismo objetivo explicado anteriormente.

Para la comunicación del lenguaje Java con Prolog se utiliza la biblioteca jpl la cual es brindada por el IDE SWI-Prolog. "*La misma presentó una aproximación a mantener un sistema consolidado para almacenar las respuestas de las consultas bajo la lógica no monótona"* (43).

Para el desarrollo del agente que tendrá integrado el sistema se estudió el modelo BDI, el cual se explica a continuación.

2.2.1 Modelo arquitectónico

El modelo Beliefs Desires Intentions (BDI) brinda un diseño para el desarrollo de un agente. Este está basado en el modelo cognitivo del ser humano, de modo que los agentes utilizan una representación interna de su entorno, un modelo del mundo que los rodea.

Un agente percibe a través de sus sensores una serie de estímulos procedentes de su entorno, los cuales modifican el modelo del mundo que tiene el agente o sea sus creencias (Beliefs) acerca del mundo. El agente tiene que tomar decisiones para interactuar con el entorno, las cuales estarán basadas en sus creencias y gobernadas por sus deseos (Desires). Estos deseos se interpretan como objetivos que el agente quiere alcanzar y ha de valerse de una serie de intenciones (Intentions) para lograrlo. Por tanto, las intenciones no son otra cosa que acciones que el agente va realizando, las cuales podrían ser abortadas en cualquier momento si las creencias del agente cambiasen.

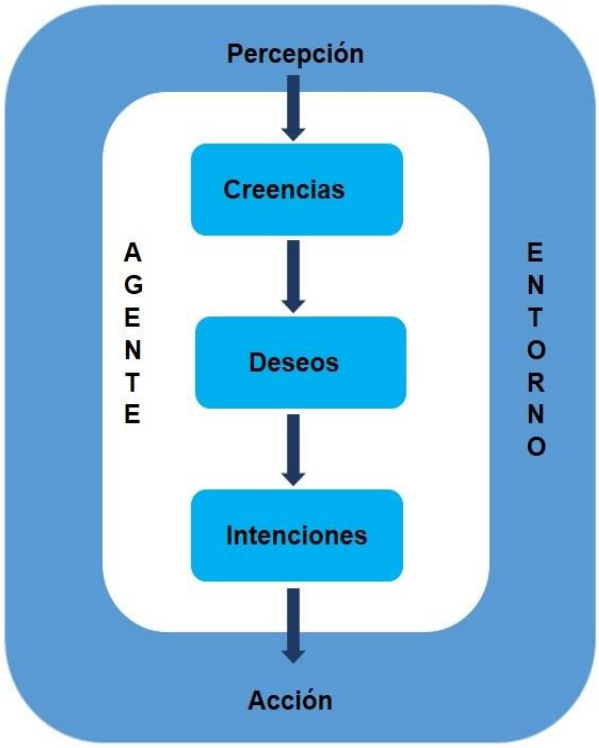


Ilustración 4. *Modelo BDI.*

El agente desarrollado para el sistema propuesto basa sus creencias en los predicados predefinidos por dicho sistema y en las percepciones que realiza en su entorno, la construcción de la base de reglas y la base de casos, la inferencia en el módulo basado en reglas y la clasificación de un nuevo caso en el módulo basado en casos.

Para cumplir con sus deseos, los cuales son sus objetivos; este debe valerse de intenciones las cuales no son más que las acciones que realizará el agente en su entorno, las cuales son: llevar a cabo la inferencia y mostrar mensajes de información al usuario. Además forman parte de sus intenciones cada uno de los predicados intermedios que se implementaron para hacer la inferencia.

Luego de haber estudiado este modelo se definió una estructura general del sistema basándose en el mismo. La misma se muestra a continuación en la ***Ilustración 4.***

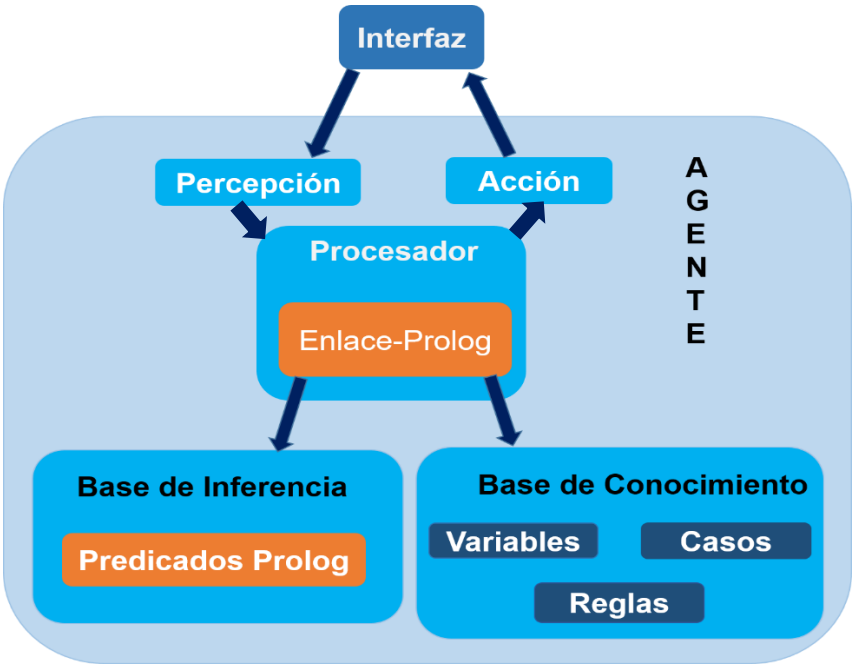


Ilustración 5. *Estructura del sistema.*

Como se aprecia en la anterior ilustración, el sistema cuenta con una capa de interfaz la cual será la que permita la interacción del usuario con el mismo. Esta capa se comunica con el agente a través de la clase Percepción, indicándole a la misma la acción que desea realizar el usuario. Luego, la clase Procesador se encarga de comunicarse con Prolog para llevar a cabo esta acción en caso de que lo requiera. Si este es el caso, el mismo se comunica con la base de inferencia para consultar los predicados predefinidos en el sistema para la inferencia de reglas y la clasificación de casos. Además, se comunica con la BC actual para consultar las variables, casos y reglas definidas en la misma. Luego de tener el resultado de la acción realizada el sistema le informa esto al usuario mediante la clase Acción.

2.3 Especificación de los requisitos de software

El presente epígrafe se refiere a la especificación de las condiciones o capacidades que el sistema debe cumplir y las restricciones bajo las cuales debe operar, logrando un entendimiento entre el equipo de desarrollo y el cliente, especificando las necesidades reales de forma que cumpla con sus expectativas.

2.3.1 Requisitos no funcionales:

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. En muchos casos los requisitos no funcionales son fundamentales en el éxito del producto. Normalmente están

vinculados a requisitos funcionales, es decir, una vez se conozca lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser (44).

Rendimiento

- El sistema debe responder en tiempo real a las solicitudes del usuario. En el caso de la inferencia y la clasificación el sistema debe demorar a lo sumo 5 segundos.

Restricciones del diseño

- Emplear como lenguaje de programación JavaFX para la interfaz visual y Prolog para el razonamiento lógico del sistema. Para dar estilo a la interfaz se utilizará Material Design, la cual es una tendencia de diseño muy utilizada actualmente en el mundo.
- El usuario debe contar con la posibilidad de modificar el código de Prolog definido en el sistema.

Apariencia o interfaz externa

- El sistema debe contar con una interfaz fácil, amigable y sencilla, permitiendo que los usuarios finales del mismo sean capaces de interactuar con este sin problema. En la interfaz principal del sistema, en el centro se encontrará en logo del mismo y en la parte inferior, las opciones: BC recientes, la ayuda e información sobre el mismo. La aplicación contará además con un menú en la parte superior mostrando las distintas acciones que se pueden realizar en el sistema, tales como: Crear una nueva BC, Abrir BC, Guardar BC, Realizar consulta a la BC, Editar los datos de la misma y Personalizar el sistema. En las interfaces de gestionar variables, casos y reglas se mostrará en la parte superior los campos requeridos para adicionar los datos deseados; en la parte central se mostrarán los datos insertados con la opción de modificar y eliminar los mismos en caso de que el usuario lo desee.
- Interfaces uniformes y con los mismos colores y diseños.
- Se debe garantizar que los colores de interfaz de la aplicación sean claros. El color que predominará será el #018DCE para el banner y los botones de acción por defecto; y el color principal será el #FFFFFF.

Portabilidad

- El sistema será capaz de correr en diferentes plataformas (Windows y Linux).

Usabilidad

- El sistema solo podrá ser usado por personas con conocimientos en los temas de IA. El software tendrá siempre visible la opción de Ayuda, lo que posibilitará un mejor aprovechamiento por parte de los usuarios de sus funcionalidades. El tiempo de aprendizaje del sistema por un usuario deberá ser menor a 20 minutos. El sistema debe proporcionar mensajes de error que sean informativos y orientados a usuario final.

Accesibilidad

- El sistema permite acceder a las funciones que brinda el mismo mediante combinaciones de teclas, las cuales son:

  Ctrl + n para crear una nueva BC.

  Ctrl + a para abrir una BC.

  Ctrl + g para guardar una BC.

  Ctrl + Shift + g para ¨Guardar como…¨

  Ctrl + Shift + v para editar las variables.

  Ctrl + Shift + r para editar las reglas.

  Ctrl + Shift + c para editar los casos.

  Ctrl + Shift + b para acceder a los módulos de consulta a la BC.

2.4 Fase I: Fase de exploración

La primera fase de XP es la de exploración, en esta los clientes definen las historias de usuario que son un artefacto de interés para la primera entrega del producto. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

2.4.1 Historias de Usuario.

Las Historias de Usuario (HU) son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Permiten, además, responder

rápidamente a los requisitos cambiantes y la sustitución de los documentos de especificación y casos de uso.

Elementos de las HU:

Número: Número asignado a la HU.

Nombre de HU: Atributo que contiene el nombre de la HU.

Fecha: Fecha en la cual fue redactada la HU.

Usuario: El usuario del sistema que utiliza o protagoniza la HU.

Prioridad en el negocio: Contiene el nivel de prioridad de la HU en el negocio. Es Alta en caso de que la HU sea indispensable en el negocio, Media en caso de que su realización no afecte el negocio y Baja cuando no se considera una prioridad.

Riesgo de desarrollo: Contiene el nivel de riesgo en caso de no realizarse la HU. Es Alta, si el riesgo de no realizar la HU incide en el funcionamiento de la plataforma, Media si el riesgo de no realizarla es medianamente importante y Baja en caso de que no se considere un riesgo tardar en la realización de la HU y no incida en el funcionamiento de la plataforma.

Puntos estimados: Este atributo es una estimación hecha por el equipo de desarrollo sobre el tiempo de duración de la HU. Cuando el valor es 1 equivale a una semana ideal de trabajo y un día equivale a 0.2 puntos.

Iteración asignada: Especifica la iteración a la que pertenece la HU correspondiente.

Descripción: Posee una breve descripción de lo que realizará la HU.

Observaciones: Aquellos detalles relevantes que serán resueltos tras la conversación del equipo desarrollador con el cliente.

A continuación se muestran algunas de las HU más significativas en el desarrollo del sistema. Consultar **Anexo 1** para ver las HU restantes.

Tabla 2. *HU Adicionar variable.*

Historia de Usuario

Número: 1	**Nombre de Historia de Usuario:** Adicionar variable
Modificación de Historia de Usuario Número:	
Usuario: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 3
Riesgo en Desarrollo: Alto	**Puntos Reales:** 2
Programador Responsable: Leduan Rosell Acosta	
Descripción: Inicia cuando el usuario decide adicionar una variable. El Sistema debe ser capaz de mostrar los campos de entrada necesarios para realizar dicha operación, los cuales son: nombre, tipo y dominio de la variable. Una vez introducidos los datos el sistema inserta la nueva variable.	
Observaciones: • Si el usuario no ha llenado todos los datos (nombre, tipo, dominio), el sistema muestra un mensaje comunicándoselo.	

Tabla 3. *HU Modificar caso*

Historia de Usuario	
Número: 5	**Nombre de Historia de Usuario:** Modificar caso
Modificación de Historia de Usuario Número:	
Usuario: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 3
Riesgo en Desarrollo: Alto	**Puntos Reales:** 3
Programador Responsable: Leduan Rosell Acosta	
Descripción: Inicia cuando el usuario decide modificar los datos de un caso, los cuales dependerán de las variables definidas en la BC.	
Observaciones:	

- El sistema debe ser capaz de mostrar un mensaje de error en caso de que el usuario entre un dato fuera de rango.

__Tabla 4.__ HU Eliminar regla.

Historia de Usuario	
Número: 9	**Nombre de Historia de Usuario:** Eliminar regla
Modificación de Historia de Usuario Número:	
Usuario: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 3
Riesgo en Desarrollo: Alto	**Puntos Reales:** 2
Programador Responsable: Leduan Rosell Acosta	
Descripción: Inicia cuando el usuario decide eliminar una o todas las reglas.	
Observaciones: • Si se desea eliminar una sola regla el sistema la elimina sin pedir confirmación. En caso de elegir la opción **Eliminar todo**, el sistema debe ser capaz de mostrar un mensaje de confirmación.	

__Tabla 5.__ HU Abrir BC.

Historia de Usuario	
Número: 11	**Nombre de Historia de Usuario:** Abrir BC
Modificación de Historia de Usuario Número:	
Usuario: Usuario	**Iteración Asignada:** 2
Prioridad en Negocio: Alta	**Puntos Estimados:** 1
Riesgo en Desarrollo: Alto	**Puntos Reales:** 0.6
Programador Responsable: Arletis Velázquez Ramírez	

Descripción: Inicia cuando el usuario decide abrir un proyecto. El sistema debe ser capaz de mostrar una ventana donde el usuario pueda seleccionar el archivo deseado, el cual debe ser del tipo .jpl y debe contener los datos necesarios de la BC.
Observaciones: • Si el archivo seleccionado no tiene la estructura adecuada, el sistema muestra un mensaje de error.

2.5 Fase II: Planificación

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Las historias generalmente valen de 1 a 3 puntos. La planificación se puede realizar basándose en el tiempo o el alcance. (45)

2.5.1 Plan de entrega

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea desplegado. Al mismo tiempo, se deben tomar precauciones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

Luego de haber realizado la fase de planificación y la de iteraciones, es necesario crear un plan de entregas que defina una fecha límite para que cada iteración sea entregada al usuario final.

Tabla 6. *Plan de entrega de versiones.*

Historia de Usuario	Final Iteración 1 (20 de febrero)	Final Iteración 2 (13 de marzo)	Final Iteración 3 (3 de abril)
Adicionar variable.	V1.0		
Modificar variable.	V1.0		
Eliminar variable.	V1.0		

Adicionar caso.	V1.0		
Modificar caso.	V1.0		
Eliminar caso.	V1.0		
Adicionar regla.	V1.0		
Modificar regla.	V1.0		
Eliminar regla.	V1.0		
Abrir BC.		V1.1	
Crear nueva BC.		V1.1	
Guardar BC.		V1.1	
Guardar BC como.		V1.1	
Consultar BC.			V1.2

2.6 Fase III: Iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores.

2.6.1 Patrones de diseño

Un patrón de diseño es una solución repetible a un problema recurrente en el diseño de software. Los patrones documentan y explican problemas de diseño y luego discuten una buena solución a dicho problema. Con el tiempo, los patrones comienzan a incorporarse al conocimiento y experiencia colectiva de la industria del software, lo que demuestra que el origen de los mismos radica en la práctica misma más que en la teoría (45).

Patrones GRASP (*General Responsibility Assignment Software Patterns*): Estos describen los principios fundamentales para la asignación de responsabilidades de cada clase que interviene en el sistema, todo esto representado mediante patrones.

Se evidencia el uso del patrón **Bajo Acoplamiento** y **Alta Cohesión**, debido a que las clases fueron diseñadas de tal forma que existiera la menor relación posible entre ellas, a la vez logrando que la información almacenada en cada clase fuera coherente. Con este diseño se logra que de existir una modificación tenga la mínima repercusión posible en el resto de las clases, se garantiza la reutilización de código y disminuye la dependencia entre clases.

Se utiliza el patrón **Controlador** en la clase MainWindowController la cual es la encargada de la separación de la lógica del negocio de la capa de aplicación, aumentando así la reutilización de código. Esta clase es la encargada de controlar todo el proceso de negocio asignando las responsabilidades a las clases correspondientes. Además controla desde la interfaz hasta el agente y le va indicando al agente los cambios en el entorno.

Se evidencia el patrón **Experto** en la clase Agente la cual es la encargada de realizar todas las operaciones relacionadas al razonamiento del sistema.

Patrones GOF (Gang of Four)

Se utiliza el patrón **Singleton** para garantizar la existencia de una única instancia de la clase SMA_Agente y la creación de un mecanismo de acceso global a dicha instancia.

Se evidencia el patrón **Strategy** el cual permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución. Se utiliza en cada una de las clases que conforman el SMA_Agente.

Tabla 7.*Plan de iteraciones.*

Iteración	Descripción	Orden de la HU a implementar	Duración total
1	En la siguiente iteración nos concentraremos en implementar las HU que tienen prioridad Alta, las cuales no son menos importantes pero no inciden	1-9	21.0

	críticamente en el funcionamiento del sistema.		
2	En esta iteración se implementarán las HU que cuentan con una prioridad Media, al igual que las anteriores son de gran importancia en cuanto a su incidencia en el funcionamiento del producto final pero muchas dependen de la implementación de las HU anteriores.	10-14	17.0
3	En esta última iteración se implementarán las HU que cuentan con una prioridad Baja, al igual que las anteriores son de gran importancia en cuanto a su incidencia en el funcionamiento del producto final pero otras HU no dependen de ellas para su implementación.	15	21.0

2.6.2 Tarjetas CRC (*Class-Responsibility-Collaboration*)

La metodología XP no requiere de la representación del sistema mediante diagramas de clases utilizando UML para diseñar las aplicaciones, en este caso utiliza otras técnicas como las tarjetas CRC (Cargo o clase, Responsabilidad, Colaboración).

La utilización de tarjetas CRC es una técnica de diseño orientado a objetos propuesta por Kent y Ward Cunningham. El objetivo de la misma es hacer, mediante tarjetas, un inventario de las clases que se van a necesitar para implementar el sistema y la forma en que van a interactuar, de esta forma se pretende facilitar el análisis y discusión de las mismas por parte de varios actores del equipo de proyecto con el objeto de que el diseño sea lo más simple posible verificando las especificaciones del sistema (46).

Tabla 8. *Tarjeta CRC MainWindowController.*

Nombre de la clase: MainWindowController

Responsabilidades	Colaborador
• Crear nueva BC. • Abrir BC. • Guardar BC.	• Agente. • Gestionar caso. • Gestionar regla. • Gestionar variable.

Tabla 9. *Tarjeta CRC Agente.*

Nombre de la clase: Agente	
Responsabilidades	**Colaborador**
• Gestionar todo el acceso a la BC y el procesamiento de la misma.	• SMA_Percepcion. • SMA_Procesador. • SMA_Accion.

2.7 Fase IV: Producción

Para que el sistema se mantenga en funcionamiento al mismo tiempo que se desarrollan nuevas iteraciones se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede disminuir después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura (45).

2.8 Consideraciones finales del capítulo

En este capítulo se dio una propuesta de solución que se ajustará a la situación problemática existente. Además las HU permitirán una mejor descripción de los requisitos con los que debe cumplir el sistema. Los requisitos del sistema y la arquitectura propuesta permitieron obtener el diseño de la aplicación. También se seleccionaron los patrones de diseño y el modelo arquitectónico BDI, los cuales optimizarán y harán más flexibles la implementación de la aplicación.

Capítulo #3: Implementación y prueba

3.1 Introducción

En este capítulo se analiza la propuesta de solución desde el punto de vista de la calidad, para evaluar cómo se le da cumplimiento al problema de la presente investigación y verificar si fueron implementadas de forma correcta cada una de las HU definidas por el cliente, además de que el sistema se esté implementando correctamente y que cuenta con las funcionalidades descritas en el capítulo anterior. Por último llevan a cabo las pruebas de Aceptación y Carga y Estrés con el objetivo de certificar que el sistema propuesto funcione de forma correcta.

3.2 Tareas de Ingeniería

Las Tareas de Ingeniería (TI) tienen como objetivo definir cada una de las actividades que dan cumplimiento a cada HU y se describen de forma clara y simple; de forma tal que se entienda lo que el sistema tiene que hacer y facilite su construcción. Los puntos estimados se transformaron en días para hacer más cómodo el proceso.

Tabla 10. *Tarea de la ingeniería 1.*

Tarea de Ingeniería	
Número de Tarea: 1	**Número de Historia de Usuario:** 1
Nombre de la Tarea: Desarrollo de la interfaz para adicionar variable.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 5 días
Fecha de Inicio: 15/12/2015	**Fecha de Fin:** 20/12/2015
Programador Responsable: Leduan B. Rosell Acosta.	
Descripción: Inicia llevando a cabo el diseño e implementación de la interfaz de adicionar variable, adaptando los componentes visuales específicos para este caso. Se debe llevar a cabo una correspondencia entre los datos entrados y los especificados por el cliente.	

Tabla 11. *Tarea de la ingeniería 2.*

Tarea de Ingeniería

Número de Tarea: 2	**Número de Historia de Usuario:** 8
Nombre de la Tarea: Implementar modificar casos.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 7 días
Fecha de Inicio: 15/2/2016	**Fecha de Fin:** 22/2/2016
Programador Responsable: Arletis Velázquez Ramírez.	
Descripción: Se implementan las funcionalidades necesarias para que el usuario pueda modificar los datos de los casos, si así lo desea.	

3.3 Estilos de programación

Un estilo de programación es un término que describe convenciones para escribir código fuente en ciertos lenguajes de programación, además de brindarle una mayor legibilidad al código. En la implementación de la aplicación se utilizaron diferentes estilos que se describen de la manera siguiente:

3.3.1 Definición de clases

La definición de clases comienza siempre con el nombre del paquete en el que se encuentra, seguido por un guión bajo y el nombre de la misma con letra inicial mayúscula.

```
public class SMA_Agente {

    public static SMA_Accion sma_accion;
    public static SMA_Percepcion sma_percepcion;
    public static SMA_Procesador sma_procesador;

    public static ObservableList<ObservableList> v_Listas;
    public static ObservableList<File> v_listaDeBC;
```

Ilustración 6. *Definición de clases.*

3.3.2 Definición de métodos

Los métodos de las clases están precedidos por la palabra reservada que define el encapsulamiento del método (public, protected o private), seguido del nombre del método el cual iniciará con letra m minúscula seguida de un guion bajo y el nombre (el cual iniciará con letra mayúscula en caso de ser una sola palabra, pero si es una palabra compuesta comenzará con minúscula la primera palabra y las demás con mayúscula).

```
public SMA_Agente() {
    sma_accion = new SMA_Accion();
    sma_percepcion = new SMA_Percepcion();
    sma_procesador = new SMA_Procesador();
}
```

Ilustración 7. *Definición de métodos.*

3.3.3 Asignaciones a variables

Se realiza siempre dejando un espacio antes y después del signo igual (=), como se muestra a continuación:

```
ObservableList<String> v_listaDeValores = observableArrayList();
ArrayList v_posVarsReales = new ArrayList();
ArrayList v_posVarsReales_CB = new ArrayList();
```

Ilustración 8. *Asignación de variables.*

3.3.4 Estructuras de control

Las estructuras de control tales como if(), for(), while(), switch(), entre otras se utilizaron colocando la llave de apertura en la misma línea de código y la de cierre en la última línea del módulo, como se puede observar a continuación:

```
for (int i = 0; i < Condicion.length; i++) {
    Term v_term = Condicion[i];
    if (v_term.toString().equals("'Y'")) {
        v_regla += " y";
    } else if (v_term.toString().equals("'O'")) {
        v_regla += " o";
    } else if (v_term.toString().equals("'NO'")) {
        v_regla += " no";
    } else if (v_term.isFloat() || v_term.isInteger()) {
        v_regla += " " + v_term.toString();
    } else {
        v_regla += " " + v_term.toString().substring(1);
    }
}
```

Ilustración 9. *Estructuras de control.*

3.4 Pruebas del sistema

Las pruebas son consideradas una de las fases más importantes en todo proyecto que se lleve a cabo, ya que estas permiten corregir la mayor cantidad de errores posibles y entregar un producto de mayor calidad. Se realizaron dos tipos de pruebas: unitarias y de aceptación. Dentro de las unitarias se llevó a cabo el método de caja blanca con la

técnica de camino básico. Para las pruebas de aceptación se realizó el método de caja negra con la técnica de partición de equivalencia.

3.4.1 Método de Caja Blanca

Es un método de prueba de software que se realiza sobre las funciones internas de un módulo. Se llevan a cabo en primer lugar, sobre un módulo concreto, para luego realizar las de caja negra sobre varios subsistemas (integración). Están centradas en el código y le permiten al desarrollador comprobar si las funciones o algoritmos implementados en el software tienen la calidad requerida o si tienen algún error. Puede proveer la solución a cualquier desperfecto que se encuentre en el código a la hora de probar, o reportar a los desarrolladores la solución al desperfecto y no solo la existencia del mismo.

El valor calculado como **complejidad ciclomática** define el número de caminos independientes del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez.

- El número de regiones del grafo de flujo coincide con la complejidad ciclomática.
    - ✓ La complejidad ciclomática, *V(G)*, se define como:

V(G) = A - N + 2

donde: *A* es el número de aristas del grafo y *N* es el número de nodos.

- ✓ La complejidad ciclomática, *V(G)*, también se define como:

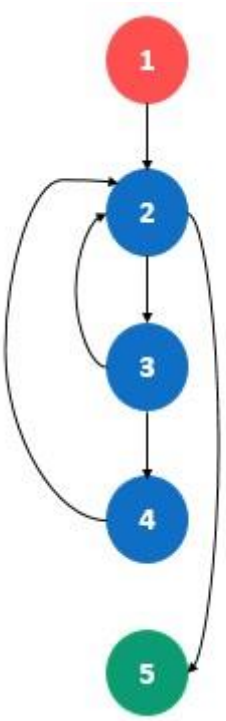
V(G) = P + 1

donde: *P* es el número de nodos predicado contenidos en el grafo *G*.

Luego de un breve análisis de la complejidad ciclomática se da paso a realizar las pruebas de Caja Blanca a los métodos de más importancia para el cliente con el fin de comprobar si tienen algún error o no cumplen con lo especificado por el cliente.

Tabla 12. *Prueba de Caja Blanca al método m_buscarModa.*

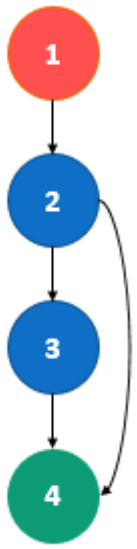
Método: m_buscarModa	**Grafo Resultante:**

private String m_buscarModa(Map<String,Integer> map) {
(1) int v_mayor = 0;
(1) String v_moda = "";
(2) for (Map.Entry<String, Integer> entry : map.entrySet())
{
(3) if(entry.getValue() > v_mayor) {
(4) v_moda = entry.getKey();
(4) v_mayor = entry.getValue();
}
}
(5) return v_moda;
}
	

Complejidad Ciclomática:	**Caminos Básicos:**
V (G)= # de regiones = 3 V (G)=A-N+2 = 6-5+2 = 3 V (G)= P+1= 2+1 = 3	{1,2,5} {1,2,3,2,5} {1,2,3,4,2,5}

Tabla 13.*Prueba de Caja Blanca al método m_Predecir.*

Método: m_Predecir	**Grafo Resultante:**

public String m_Predecir(ArrayList<String> v_listaHechos, String v_probar) {
(1) v_query = new Query("predecir(" + v_listaHechos + "," + v_probar + ",Certeza).");
(2) if (v_query.hasSolution()) {
(3) float v_certeza = v_query.oneSolution().get("Certeza").floatValue();
(3) return String.valueOf(v_certeza);
}
(4) return "-2";
}
	

Complejidad Ciclomática:	**Caminos Básicos:**
V (G)= # de regiones = 2 V (G)=A-N+2 = 4-4+2 = 2 V (G)= P+1= 1+1 = 3	{1,2,4} {1,2,3,4}

Tras haber identificado los casos de prueba de Caja Blanca se le aplicaron las pruebas a cada camino identificado de cada caso de prueba. Para definir los casos de prueba es necesario tener en cuenta:

- **Descripción:** Se describe el caso de prueba y se especifican los aspectos fundamentales de los datos de entrada.
- **Condición de ejecución:** Se verifica que cada parámetro cumpla las condiciones de ejecución.
- **Entrada:** Se muestran los parámetros de entrada del procedimiento.
- **Resultados Esperados:** Se especifica el resultado que debe arrojar el procedimiento.

Tabla 14. *Caso de Prueba para camino básico #1.*

Caso de Prueba Camino Básico #1
Descripción: Se realiza la predicción de la certeza del hecho introducido por el usuario.
Condiciones: Se debe haber cargado el fichero en la aplicación y acceder a la vista

del módulo SBR.
Entrada: Nota: En la BC utilizada se definieron las siguientes reglas: • si vLeduan o no vArletis y vFemenino entonces vUCI con_certeza 0.9. • si vMasculino o 23.0 entonces vLeduan con_certeza 0.5. • si vFemenino y vArletis entonces 23.0 con_certeza 0.9. v_listaHechos: [0.9:vFemenino, 0.7:vArletis] v_probar: 23
Resultado Esperado: El sistema debe emitir un mensaje mostrando la certeza con que se cumple el hecho “23”, la cual debe ser 0.63.
Resultado: Satisfactorio.

Tabla 15. *Caso de Prueba camino básico #2.*

Caso de Prueba Camino Básico #2
Descripción: Se realiza la predicción de la certeza del hecho introducido por el usuario.
Condiciones: Se debe haber cargado el fichero en la aplicación y acceder a la vista del módulo SBR.
Entrada: Nota: En la BC utilizada se definieron las siguientes reglas: • si vLeduan o no vArletis y vFemenino entonces vUCI con_certeza 0.9. • si vMasculino o 23.0 entonces vLeduan con_certeza 0.5. • si vFemenino y vArletis entonces 23.0 con_certeza 0.9. v_listaHechos: [0.7:vArletis] v_probar: vLeduan
Resultado Esperado: El sistema debe emitir un mensaje indicando que no se puede

probar la veracidad del hecho “vLeduan”.
Resultado: Satisfactorio.

Resultados: Se probaron, mediante casos de pruebas de caja blanca o Pruebas Unitarias (PU) un total de 18 funcionalidades del sistema de las cuales 12 fueron satisfactorias, representando el 66,6 %. Las demás funcionalidades fueron corregidas en una segunda y tercera iteración.

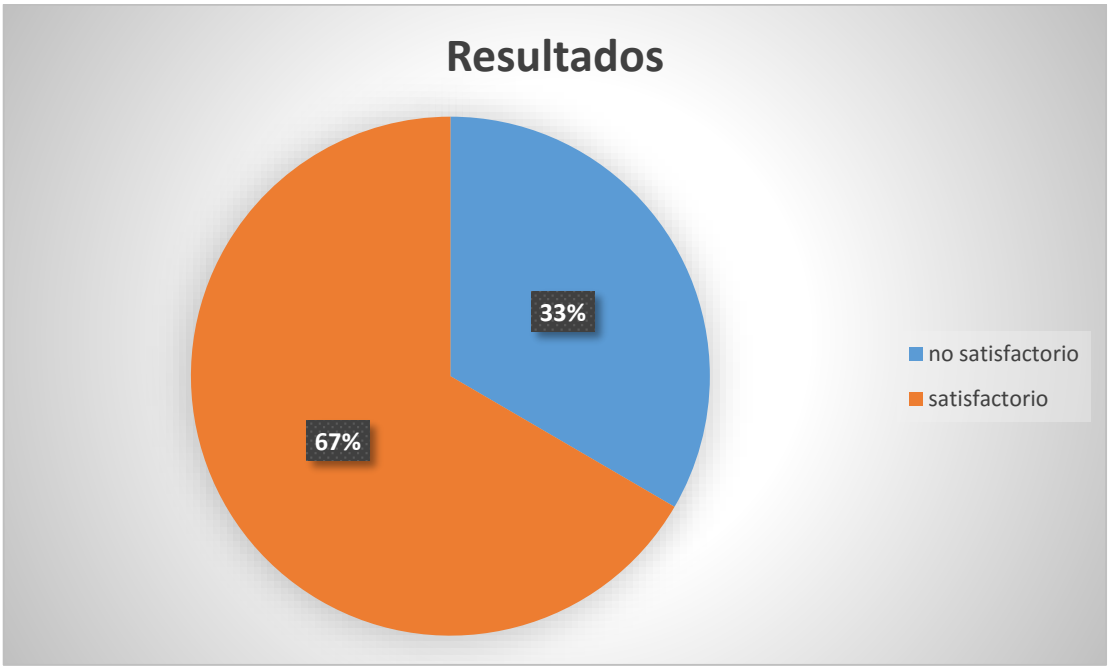


Ilustración 10. *Resultados de pruebas de Caja Blanca.*

3.4.2 Pruebas de Aceptación

En esta prueba se evalúa el grado de calidad del software con relación a todos los aspectos relevantes para que el uso del producto se justifique. Son básicamente pruebas funcionales, sobre el sistema completo y buscan una cobertura de la especificación de requisitos y del manual del usuario. Estas pruebas no se realizan durante el desarrollo, pues sería impresentable al cliente; sino que se realizan sobre el producto terminado e integrado o pudiera ser una versión del producto o una iteración pactada previamente con el cliente.

Tabla 16. *Caso de Prueba Adicionar variable.*

Caso de Prueba de Aceptación

Código: HU1_P1	**Historia de Usuario:** 1
Nombre: Adicionar variable.	
Descripción: Esta prueba se encarga de validar la HU “Adicionar variable”, adicionando a la BC la nueva variable.	
Condiciones de Ejecución: El usuario debe haber llenado todos los campos requeridos (nombre, tipo, dominio).	
Entradas/Pasos de Ejecución: 1. Ir al botón Expandir Menú. 2. Luego de que el sistema muestre el Menú, hacer click en Nueva BC. 3. Luego el sistema muestra una ventana para que el usuario seleccione la ubicación donde desea guardar la nueva BC. 4. Inmediatamente después el usuario es dirigido a la vista Gestionar variables. 5. Luego se introduce en el dato nombre el valor “sexo”, como tipo “nominal” y en el campo dominio se introducen los valores “masculino, femenino”. 6. Hacer clic en el botón Adicionar.	
Resultado Esperado: El sistema debe adicionar en la BC la nueva variable.	
Resultado Obtenido: La variable fue adicionada correctamente.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 17. *Caso de Prueba Abrir BC.*

Caso de Prueba de Aceptación	
Código: HU11_P1	**Historia de Usuario:** 11
Nombre: Abrir BC.	
Descripción: Esta prueba se encarga de validar la HU “Abrir BC”, abriendo la BC seleccionada.	
Condiciones de Ejecución:	
Entradas/Pasos de Ejecución: 1. Ir al botón Expandir Menú. 2. Luego de que el sistema muestre el Menú, hacer click en Abrir BC. 3. Luego el sistema muestra una ventana para que el usuario seleccione la ubicación donde se encuentra la BC que desea abrir. 4. Inmediatamente después el sistema muestra el siguiente mensaje “La(s) BC(s) ha(n) sido cargada(s) satisfactoriamente”. 5. Luego se habilitan las opciones: Guardar BC, Guardar BC como…, Editar… y Consultar BC para que el usuario seleccione la acción que desea realizar a continuación.	
Resultado Esperado: El sistema debe abrir la BC seleccionada.	
Resultado Obtenido: La BC fue abierta correctamente.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 18. *Caso de Prueba Modificar regla.*

Caso de Prueba de Aceptación	
Código: HU8_P1	**Historia de Usuario:** 8
Nombre: Modificar regla.	
Descripción: Esta prueba se encarga de validar la HU “Modificar regla”, modificando en la BC la regla con sus nuevos datos.	
Condiciones de Ejecución: La regla que se desea modificar tiene que estar añadida	

en la BC.
Entradas/Pasos de Ejecución: 1. Ir al botón Expandir Menú. 2. Luego de que el sistema muestre el Menú, hacer click en Editar reglas. 3. Luego el sistema muestra un menú desplegable para el que el usuario seleccione la BC de la cual desea editar sus reglas. 4. Luego el usuario selecciona la BC “prueba1.jpl”. 5. Inmediatamente después el sistema muestra la vista Gestionar reglas. 6. Luego el usuario da click en el botón Modificar regla de la regla “si Arletis y femenino entonces 23 con_certeza 0.9”. 7. Posteriormente el sistema muestra los campos a modificar de la regla anteriormente seleccionada, 8. Luego de modificar los datos deseados, el usuario da click en Modificar.
Resultado Esperado: El sistema debe modificar la regla correctamente.
Resultado Obtenido: La regla fue modificada correctamente.
Evaluación de la Prueba: Satisfactoria.

Resultados: De un total de 15 casos de PA, 2 de ellos resultaron no satisfactorios, lo cual representa el 13.3% del total de casos de pruebas realizadas. Mientras que los 13 restantes resultaron satisfactorios para un 86.7% *(Figura 17)*. Los errores detectados por los casos de pruebas no satisfactorios fueron mitigados después de 3 iteraciones de prueba.

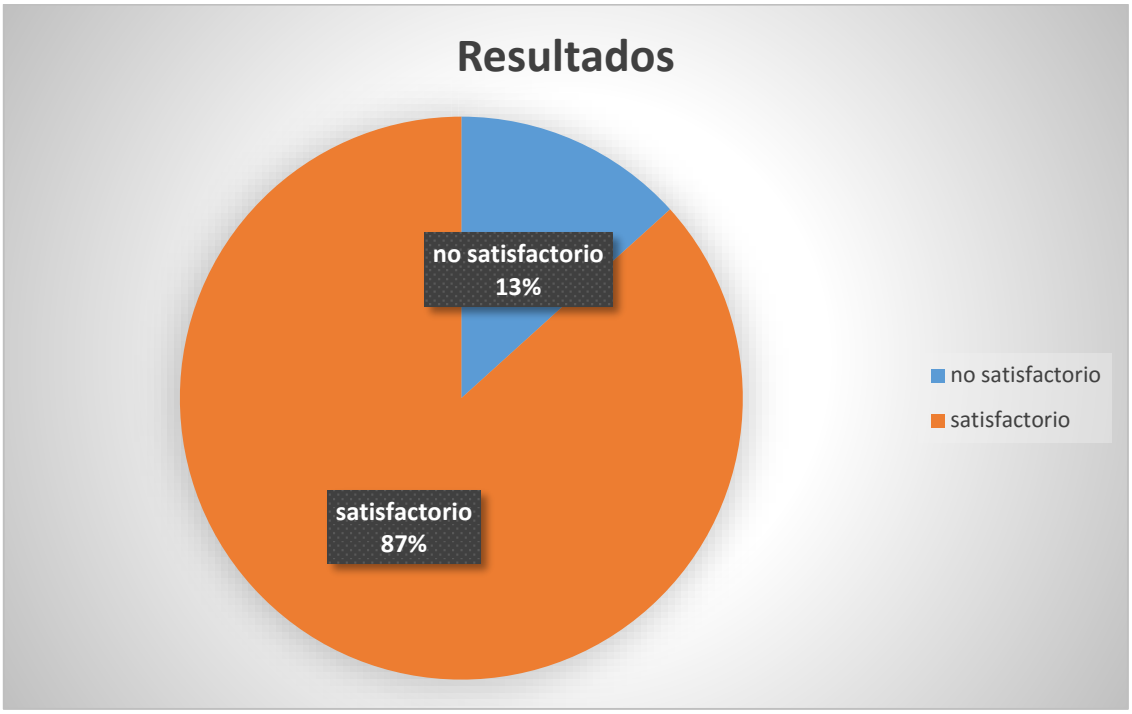


Ilustración 11.*Resultados de pruebas de Aceptación.*

3.5 Consideraciones finales del capítulo

Se definieron las tareas de ingeniería las cuales permitieron evitar la sobrecarga de trabajo en el equipo de desarrollo y agilizar aún más dicho proceso. La aplicación de las pruebas de integración permitió validar la integridad del código, mientras que las pruebas de aceptación posibilitaron comprobar la correcta implementación de las historias de usuarios definidas con anterioridad. Además se definieron los diferentes estilos de programación presentes en el sistema.

Conclusiones

Con la realización de este trabajo se desarrolló un Sistema Inteligente para la resolución práctica de ejercicios en la disciplina Inteligencia Artificial utilizando la programación lógica. De esta forma se da cumplimiento al objetivo propuesto al inicio de la investigación, además se comprobó que:

- La elaboración del marco teórico permitió corroborar la vigencia de las investigaciones y el desarrollo de sistemas inteligentes y la programación lógica, lo cual permitió conformar una base teórica para el desarrollo del sistema propuesto.
- A pesar de existir varias metodologías específicas para el desarrollo de sistemas agentes, las facilidades de la metodología ágil XP permitió conducir de forma acertada el desarrollo de la solución propuesta.
- Se obtuvo un agente lógico con un módulo de razonamiento basado en reglas y uno basado en casos. En este último el agente es capaz de calcular, a partir de sus creencias, el peso de los rasgos y un grado de certeza probabilista para los valores de los rasgos en cada caso que conforma la base de conocimiento.
- Las pruebas realizadas al sistema permitieron apreciar el cumplimiento de los requisitos exigidos inicialmente y la validación de la solución propuesta mediante la satisfacción del cliente.

Recomendaciones

Después de la investigación realizada y los resultados obtenidos durante la creación de la aplicación, se recomienda:

- El despliegue de la aplicación en la universidad para que pueda ser utilizada por todos los estudiantes.
- Incorporar nuevas formas de calcular la certeza.

Referencias Bibliográficas

1. **Instituto Tecnológico de Aguascalientes. [En línea] 18 de 02 de 2013. [Citado el: 19 de 01 de 2016.] http://ticstecags.blogspot.com/2013/02/11-las-tics-y-areas-de-aplicacion.html.**

2. E. Castillo, J.M. Gutiérrez, and A.S. Hadi. *Sistemas Expertos y Modelos de Redes Probabilísticas.* España : Academia Española de Ingeniería, 1998.

3. Sitio oficial Universidad de Antioquia. [En línea] febrero de 2010. [Citado el: 10 de noviembre de 2015.] http://ingenieria.udea.edu.co/~jbuitrago/instrumentacionElectronica/Clases/Clase09-LogicaDifusa1.pdf.

4. Springer. *Advances on Practical Applications of Agents and Multiagent Systems.* 2011.

5. eurodecision. *Operational Research.* [En línea] 2014. [Citado el: 18 de 01 de 2016.] http://www.eurodecision.es/systemes-a-base-de-regles.

6. Espinoza R., Alfredo, Quintero R., Agustín y Venecia Zambrano, S. Instituto de Investigaciones Eléctricas. *Sistema experto basado en casos para un sistema de diagnóstico en tiempo real.* [En línea] julio-septiembre de 2016. [Citado el: 09 de febrero de 2016.] http://www.iie.org.mx/boletin032006/apli.pdf.

7. Santana Tejero, Ruth. [En línea] [Citado el: 03 de febrero de 2016.] http://www.it.uc3m.es/jvillena/irc/practicas/estudios/DENDRAL.pdf.

8. López Michelone, Manuel. UNOCERO. [En línea] 22 de abril de 2013. [Citado el: 03 de febrero de 2016.] https://www.unocero.com/2013/04/22/mycin-un-sistema-experto-asombroso-que-no-se-usa/.

9. Sistemas expertos . [En línea] 15 de marzo de 2011. [Citado el: 04 de febrero de 2016.] http://sistemasexpertosx2011.blogspot.com/2011/03/weka.html.

10. *JavaLog: un Lenguaje para la Programación de Agentes.* Zunino, Alejandro, Berdún, Luis y Amandí, Analía. 13, Buenos Aires : s.n., 2001, Revista Iberoamericana de Inteligencia Artificial., págs. 94-99. 1137-3601.

11. *Achitecture and Design of Expert System for Quality of Life Evaluation.* Atanasova, Irena y KŘupka, Jiří. No 3, Pardubice, Czech Republic : s.n., marzo de 2013, Informática Económica, Vol. vol 13. 1453-1305.

12. Beck, Kent y Andres, Cynthia. *Extreme Programming Explained. Second Edition. Embrace Change.* Boston : Addison-Wesley, 2012.

13. *A Hybrid Architecture for Web-based Expert Systems.* Dunstan , Neil . New England : s.n., 2012, International Journal of Artificial Intelligence and Expert Systems , Vol. vol. 3.

14. *An IPC-based Prolog design pattern for integrating backward chaining inference into applications or embedded system.* Guoqi, Li , y otros. No 6,

Beijing, China : s.n., 22 de octubre de 2014, Chinese Journal of Aeronautics, Vol. vol 27. 1571–1577.

15. *Development of knowledge Base Expert System for Natural treatment of Diabetes disease.* Jha, Kumar Sanjeev y Singh, D.K. No. 3, Bihar, India : s.n., 2012, International Journal of Advanced Computer Science and Applications, Vol. vol. 3.

16. *SISTEMAS EXPERTOS PARA LA ENSEÑANZA Y EL APRENDIZAJE DE LA MATEMÁTICA EN LA EDUCACIÓN SUPERIOR.* Vílchez Quesada, Enrique. No. 3, 2007, CUADERNOS DE INVESTIGACIÓN Y FORMACIÓN EN EDUCACIÓN MATEMÁTICA, págs. 45-67.

17. *Development of an Expert System for Dust Explosion Risk Management Based on ASP.Net and Prolog.* Zhang , Qian , Zhong, Shengjun y Jiang, Guanyu . No. 9, Shenyang, China : ACADEMY PUBLISHER, septiembre de 2011, JOURNAL OF SOFTWARE, Vol. VOL. 6. 1857-1865.

18. *Design and Implementation of Fuzzy Expert System for Back pain Diagnosis.* Kadhim, Abbas Mohammed, Alam, M.Afshar y Kaur, Harleen . No. 9, New Delhi, India : s.n., septiembre de 2011, INTERNATIONAL JOURNAL OF INNOVATIVE TECHNOLOGY & CREATIVE ENGINEERING, Vol. vol. 1. 2045-8711.

19. *A BELIEF REVISION SYSTEM FOR LOGIC PROGRAMS.* Taher , Ali, Najem, Ziad y Sapiyan, Mohd . [ed.] David C. Wyld. Kuwait : COSIT, DMIN, SIGL, CYBI, NMCT, AIAPP, 2014, págs. 227–231.

20. Torres Rubio, Yoannys, Cordero Morales, Dasiel y Ruiz Constanten, Yadira. Revista Cubana de Ciencias Informáticas. *Sistema de Razonamiento Basado en Casos para la identificación de riesgos de software.* [En línea] ISSN 2227-1899, abril-junio de 2013. http://rcci.uci.cu.

21. *Sistema de diagnóstico para la estimación de secciones en fallo en sistemas eléctricos de potencia.* Bravo, Marta, y otros. No. 2-3 , La Habana : s.n., septiembre de 2006, Ingeniería Energética, Vol. vol. XXVII.

22. *UN AMBIENTE INTEGRADO PARA LA ENSEÑANZA DE SISTEMAS EXPERTOS EN FORMA PARTICIPATIVA.* Lezcano Brito, Mateo y Valdés Pardo, Víctor Giraldo. No. 1, Santa Clara : s.n., Informática Educativa, Vol. vol. 11, págs. 69-81.

23. Lezcano Brito, Mateo y Valdés Pardo, Giraldo . *Logsim. Sistema de apoyo al aprendizaje de la programación.* Facultad de Matemática, Física y Computación., Universidad Central de Las Villas (UCLV). Santa Clara : s.n.

24. *Una arquitectura unificada para el razonamiento borroso. A Uniform Framework for the Fuzzy Reasoning.* Morell Pérez, Carlos y Bello Pérez, Rafael . No.4, abril de 2007, REVISTA CUBANA DE CIENCIAS INFORMÁTICAS, Vol. vol.1, págs. 68-83.

25. Peiró, Karma. Karma Peiró. *Historias de una periodista digital.* [En línea] 27 de junio de 2010. [Citado el: 03 de febrero de 2016.] http://karmapeiro.blogspot.com/2006/03/los-agentes-inteligentes-entendern-el.html.

26. Relloso, Gerardo. *Departamento de Producción de Colegial Bolivariana.* Caracas, Venezuela : Psicología. pág. 121.

27. Blackburn, Simon. *The Oxford Dictionary of Philosophy.* s.l. : Oxford University Press, 2010.

28. Russell, Stuart y Norvig, Peter. *Inteligencia Artificial: Un enfoque moderno.* 3era. págs. 274-276.

29. Commons, Creative. Portal Web de la Universidad de Murcia. [En línea] 30 de 12 de 2006. [Citado el: 15 de diciembre de 2015.] http://www.um.es/docencia/barzana/IAGP/Iagp2.html.

30. Wells, Don. Extreme Programming: A gentle introduction. [En línea] 08 de 10 de 2013. [Citado el: 12 de diciembre de 2015.] http://www.extremeprogramming.org/.

31. Morandini, M y Nguyen, D.C. *Tool-Supported Development with Tropos: The Conference Management System Case Study.* [ed.] M. Luck y L. Padgham . Toronto : s.n., 2008. págs. 182-196.

32. *Modelling and simulation of social systems with INGENIAS.* Sandores, C. y Pavón, J. 2008, International Journal of Agent-Oriented Software Engineering, págs. 196-221.

33. *Agent-Oriented Methodologies.* Giorgini, P. y Henderson-Sellers, B. Hershey - London - Melbourne - Singapore : s.n., 2005, Idea Group Publishing, pág. 420.

34. Gómez-Sanz, J. *Modelado de Sistemas Multi-Agentes.* Universidad Complutense de Madrid. Madrid : s.n., 2002. Doctoral.

35. Gómez Sanz, Jorge. *Metodologías para el desarrollo de sistemas Multi-Agente.* 2003. págs. 51-63.

36. Gómez Sanz, Jorge y Pavón Mestras, Juan. *Desarrollo de Sistemas Multi-Agentes.* s.l. : Dep. de Sistemas Informáticos y Programación, 2004.

37. INGENIAS. [En línea] 2005. [Citado el: 11 de noviembre de 2015.] http://grasia.fdi.ucm.es/ingenias..

38. Metodología XP Vs.Metodología RUP. [En línea] 3 de junio de 2016. [Citado el: 10 de junio de 2016.] http://metodologiaxpvsmetodologiarup.blogspot.com/2008/04/cuadro-comparativo-metodologia-xp-vs.html.

39. Bergin, Thomas J. y Gibson, Richard G. *History of Programming Languages II.*

40. Russell, Stuart y Norvig, Peter. *Inteligencia Artificial: Un enfoque moderno.* 3era. págs. 328-330.

41. Llorens Largo, Faraón y Castel de Haros, María Jesús. *Prácticas de Lógica Prolog.* Alicante : s.n., 2001.

42. *swi-prolog.* [En línea] 2015. [Citado el: 23 de noviembre de 2015.] www.swi-prolog.org..

43. *JPL : IMPLEMENTATION OFA PROLOG SYSTEM SUPPORTING INCREMENTAL TABULATION.* Ali, Taher, Najem, Ziad y Sapiyan, Mohd . [ed.] Jan Zizka. Kuwait : CCSIT, SIPP, AISC, CMCA, SEAS, CSITEC, DaKM, PDCTA, NeCoM, 2016, págs. 323–338.

44. *Inserción profesional de los egresados de la linceciatura en Administración Turística. .* Oliveira, M M. 2010, insercionlaboralegresados.

45. Canos, José y Penades, María Carmen. Metodologías Ágiles en el Desarrollo de Software. [En línea] 2002.

46. Jummp's Blog. jummp.wordpress. *Gestión de proyectos y desarrollo de software.* [En línea] wordpress, 10 de enero de 2012. [Citado el: 04 de febrero de 2016.] https://jummp.wordpress.com/2012/01/10/desarrollo-de-software-tarjetas-crc/.

47. *Reglas que permiten explotar el conocimiento de los esquemas de relés para la localización de la sección el fallo.* Bravo, Marta, y otros. No 2-3, La Habana : Cujae, 2006, Ingeniería Energética, Vol. vol XXVII.

Anexos 1: Historias de Usuario

Tabla 19.HU Modificar variable.

Historia de Usuario	
Número: 2	**Nombre de Historia de Usuario:** Modificar variable
Modificación de Historia de Usuario Número:	
Usuario: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 1
Riesgo en Desarrollo: Alto	**Puntos Reales:** 0.6
Programador Responsable: Arletis Velázquez Ramírez	
Descripción: Inicia cuando el usuario decide modificar una variable. El sistema debe ser capaz de permitir modificar los datos (nombre, tipo, dominio) de la misma.	
Observaciones: • En caso de que el usuario introduzca un dato incorrecto el sistema debe mostrar un mensaje de error.	

Tabla 20.HU Eliminar variable.

Historia de Usuario	
Número: 3	**Nombre de Historia de Usuario:** Eliminar variable
Modificación de Historia de Usuario Número:	
Usuario: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 1
Riesgo en Desarrollo: Alto	**Puntos Reales:** 0.8
Programador Responsable: Leduan B. Rosell Acosta	
Descripción: Inicia cuando el usuario decide eliminar una variable. El sistema debe ser capaz de mostrar los cambios realizados por el usuario.	

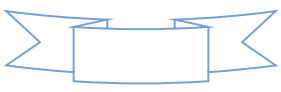
Observaciones: • Si se desea eliminar una sola variable el sistema la elimina sin pedir confirmación. En caso de elegir la opción **Eliminar todo**, el sistema debe ser capaz de mostrar un mensaje de confirmación.

Tabla 21.*HU Adicionar regla.*

Historia de Usuario	
Número: 7	**Nombre de Historia de Usuario:** Adicionar regla
Modificación de Historia de Usuario Número:	
Usuario: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 1
Riesgo en Desarrollo: Alto	**Puntos Reales:** 0.8
Programador Responsable: Leduan B. Rosell Acosta	
Descripción: Inicia cuando el usuario decide adicionar una nueva regla. El sistema debe ser capaz de mostrar los campos para que el usuario introduzca los datos que desea, los cuales dependerán de las variables definidas en la BC.	
Observaciones: • El sistema debe ser capaz de mostrar un mensaje de error en caso de que algún dato sea incorrecto o esté vacío.	

Tabla 22.*HU Modificar regla.*

Historia de Usuario	
Número: 8	**Nombre de Historia de Usuario:** Modificar regla
Modificación de Historia de Usuario Número:	
Usuario: Usuario	**Iteración Asignada:** 1

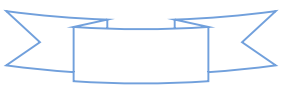


Prioridad en Negocio: Alta	**Puntos Estimados:** 1
Riesgo en Desarrollo: Alto	**Puntos Reales:** 0.6
Programador Responsable: Leduan B. Rosell Acosta	
Descripción: Inicia cuando el usuario decide modificar los datos de una regla. El sistema debe ser capaz de mostrar los campos nuevamente para que el usuario cambie los datos que desee, los cuales dependerán de las variables definidas en la BC.	
Observaciones: • El sistema debe ser capaz de mostrar un mensaje de error en caso de que algún dato sea incorrecto o esté vacío.	

Tabla 23. *Adicionar caso.*

Historia de Usuario	
Número: 4	**Nombre de Historia de Usuario:** Adicionar caso
Modificación de Historia de Usuario Número:	
Usuario: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 1
Riesgo en Desarrollo: Alto	**Puntos Reales:** 0.6
Programador Responsable: Arletis Velázquez Ramírez	
Descripción: Inicia cuando el usuario decide adicionar un nuevo caso al sistema. El sistema debe ser capaz de mostrar los campos para que el usuario introduzca los datos que desea, los cuales dependerán de las variables definidas en la BC.	
Observaciones: • El sistema debe ser capaz de mostrar un mensaje de error en caso de que algún dato sea incorrecto o esté vacío.	

Tabla 24.*Eliminar caso.*



Historia de Usuario	
Número: 6	**Nombre de Historia de Usuario:** Eliminar caso
Modificación de Historia de Usuario Número:	
Usuario: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 1
Riesgo en Desarrollo: Alto	**Puntos Reales:** 0.8
Programador Responsable: Arletis Velázquez Ramírez	
Descripción: Inicia cuando el usuario decide eliminar un caso al sistema. El sistema debe ser capaz de eliminar el caso seleccionado por el usuario.	
Observaciones: • Si se desea eliminar un solo caso, el sistema la elimina sin pedir confirmación. En caso de elegir la opción **Eliminar todo**, el sistema debe ser capaz de mostrar un mensaje de confirmación.	

Tabla 25.*HU Crear nueva BC.*

Historia de Usuario	
Número: 10	**Nombre de Historia de Usuario:** Crear nueva BC
Modificación de Historia de Usuario Número:	
Usuario: Usuario	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 1
Riesgo en Desarrollo: Alto	**Puntos Reales:** 0.8
Programador Responsable: Arletis Velázquez Ramírez	
Descripción: Inicia cuando el usuario decide crear una nueva base de conocimiento.	
Observaciones:	

- Inicialmente el sistema le pide al usuario que especifique la dirección donde se guardará la nueva BC. Una vez especificada la dirección el sistema lo dirige a la vista Gestionar Variable.

Tabla 26.*HU Guardar BC.*

Historia de Usuario	
Número: 12	**Nombre de Historia de Usuario:** Guardar BC
Modificación de Historia de Usuario Número:	
Usuario: Usuario	**Iteración Asignada:** 2
Prioridad en Negocio: Baja	**Puntos Estimados:** 1
Riesgo en Desarrollo: Bajo	**Puntos Reales:** 0.8
Programador Responsable: Leduan B. Rosell Acosta	
Descripción: Inicia cuando el usuario decide guardar una o varias BC sobre la que se ha estado trabajando.	
Observaciones:	

Anexo 2: Tarjetas CRC

Tabla 27.*Tarjeta CRC SMA_Accion*

Nombre de la clase: SMA_Accion	
Responsabilidades	**Colaborador**
• Realizar acciones en el sistema.	• Agente. • SMA_Procesador.

Tabla 28.*Tarjeta CRC SMA_Procesador*

Nombre de la clase: SMA_Procesador	
Responsabilidades	**Colaborador**
• Procesar toda la información percibida por el sistema.	• Agente. • SMA_Percepcion. • SMA_Accion. • SMA_Enlace_Prolog.

Tabla 29.*Tarjeta CRC SMA_Enlace_Prolog*

Nombre de la clase: SMA_ Enlace_Prolog	
Responsabilidades	**Colaborador**
• Comunicar el sistema con Prolog.	• SMA_Procesador. • jpl.

Tabla 30.Tarjeta CRC SMA_Percepcion

Nombre de la clase: SMA_ Percepcion	
Responsabilidades	**Colaborador**
• Percibir todos los cambios ocurridos en la interfaz del sistema.	• SMA_Procesador. • Agente.

Tabla 31.Tarjeta CRC Gestionar Caso

Nombre de la clase: Gestionar Caso.	
Responsabilidades	**Colaborador**
• Adicionar nuevo caso. • Modificar Caso. • Eliminar caso.	• MainWindowController. • SBC_Caso.

Tabla 32. Tarjeta CRC Gestionar Regla

Nombre de la clase: Gestionar Regla.	
Responsabilidades	**Colaborador**
• Adicionar nueva regla. • Modificar regla. • Eliminar regla.	• MainWindowController. • SBR_Regla.

Tabla 33. Tarjeta CRC Gestionar Variable

Nombre de la clase: Gestionar Variable.	
Responsabilidades	**Colaborador**
• Adicionar nueva variable. • Modificar variable. • Eliminar variable.	• MainWindowController. • Caux_Variable.