Universidad de las Ciencias Informáticas
Facultad 3

Rediseño e implementación de una herramienta interactiva para la simulación de procesos industriales: Nodo Virtual de Procesos.

Trabajo de Diploma para optar por el título de
Ingeniero Informático

Autor(es): Yelina Seija Rodríguez
Julio Jesús García Guevara

Tutor: MSc. Yalice Gámez Batista

Viernes, 17 de junio de 2011
Ciudad de La Habana

Para todo problema humano hay siempre una solución fácil, clara, plausible y equivocada.

Henry-Louis Mencken

AGRADECIMIENTOS

Julio:

Agradecer a todas las personas que de alguna manera me han ayudado seria casi un capítulo más de la tesis, trataré de ser lo más explicito posible, con el fin de que todos quepan en una cuartilla.

A la Msc. Yalice Gámez Batista, Vicedecana de Formación de la Facultad 3, por haber confiado en las nuevas ideas de nosotros y proveernos de un tema de investigación apasionante e interesante. Por compartir el escaso tiempo personal que le quedaba a raíz de su dedicación al trabajo.

Agradezco a mis profesores de todas las enseñanzas previas a la universitaria, profesores que siempre han tratado y de alguna manera han logrado encaminarme en la vida, agradezco de forma especial a Diusmar, ayer profesor, hoy amigo entrañable, a Heyler, el gran Mario, amigos muy buenos que nunca me vieron como un estudiante, siempre fui para ellos solo Julio.

A Daylen, mi mejor amiga, la única que ha estado conmigo en todas las enseñanzas, aunque después no separamos para ir a la Universidad, amiga entrañable de mi infancia. A Aile, a mis primos que siempre han estado para mi, siempre.

A mis amigos del "Verde", hoy algunos están aquí conmigo, otros quisieran estar pero sé que no pudieron.

A mis amigos de la Universidad, a esa pandilla de locos que año tras año agrega más personas a nuestras gloriosas filas, a Ale, Eugenio, Alexander, Emilito, la Krysia, mi hemosa hermanita Yai, el Yasse, a mi amigo Ariel, cuya rivalidad amistosa que existió

en los primeros años de nuestra carrera nos proporcionó a ambos deseos de aprender y superarnos, a Suamy, a Asdrubal, a mis buenos amigos, hermanos de cariño, Ramses el Pina y Rene. Casi todos del grupo 4102. Un grupo donde todos hemos sido más que amigos, donde hemos aprendido que sobrellevar cargas entre muchos es mejor que hacerlas solo.

A mis amigos Anilie, Daymi, Daylin, Ernesto Quinta, amigo del cual me apoye en un momento de flaqueza. Te quiero Viejo y gracias.

Al profesor Alain, cuya metodología y manera tan especial de dar las clases me ayudo en mi vocación dentro de mi carrera informática, todavía recuerdo el primer consejo que nos dio apenas llegamos en primer año y todavía teníamos en polvo del viaje como quien dice en la ropa: "Esta es una carrera de fondo, no de rapidez, los más rápidos y constantes serán los mejores".

A dos profesoras que tengo entre las más queridas de aquellas que me han dado clases: las profesoras Águeda y Ana Cecilia quien me brindó todo su apoyo y su ayuda incondicional aun estando tan lejos.

Un agradecimiento especial, muy especial a las profesoras Linnet y Yurita que han sido como hermanas mayores, en parte esta tesis es gracias al apoyo de las dos, gracias por soportarme. Muchas gracias, las quiero. Un agradecimiento a Zoraya, por tener siempre un consejo a flor de labios, y por dejarme pertenecer al grupo de amistades que rodean a nuestro hermanito menor Rafael.

Agradecimientos

A mis amigos de Venezuela, los que están y los que no, a mi tía Arelis por enseñarme algunas cosas que me ayudaron a crecer como persona, a Anibal y todos mis demás amigos de la misión.

A la coautora de este trabajo, la Ing. Yelina Seija cuya seriedad y consagración al mismo hizo posible su terminación.

A los compañeros de clases que día a día estuvieron conmigo. Un beso especial a Daylin, a mis amigos que estuvieron conmigo en Venezuela, finalmente a mi pequeño pueblito de Dos Palmas.

DEDICATORIA

Julio:

Dedico el fruto de mis horas de estudio e investigación a mi familia completa, que entre todos formamos por nuestras diferencias en cuanto a ideas y hábitos, un maravilloso ecosistema del cual soy solo una parte.

A mi padre Dr. Julio A. García Medina, un ejemplo de hombre honrado y excelente profesional, con quien aprendí que la humanidad es el rasgo más altruista que cada persona debe tener, por llevar a mi casa el milagro cotidiano de dedicar su vida a ayudar a los demás.

A mi madre Dra. Elizabeth Guevara Mora, un ejemplo de mujer sacrificada e íntegra en sus principios y sus convicciones, razones que tampoco han impedido que sea una de las personas más admiradas por su trabajo y su profesionalidad, con quien aprendí que el trabajo forma el carácter del hombre y que lo más importante para una persona debe ser y es: la familia. Cuya ternura es uno de los tesoros más grandes que poseo y que guardo con más recelo en mi corazón.

A mis abuelos que han ayudado a moldear mi niñez y me han dado, aun sin tener grandes conocimientos culturales, alas para cumplir mis sueños.

Al fruto principal de este trabajo, mis hermanos David y Cristian, camino y razón más importante para buscar la excelencia académica con el deseo de convertirme en un buen profesional, espero ser un ejemplo digno a seguir. Sé que ambos superarán incluso mis altas expectativas.

Agradecimientos

Este trabajo no habría sido posible sin el apoyo conjunto de todos ustedes, gracias.

A la Revolución, por darme lo que muchos desean tener en otras partes del mundo y no tienen: Una oportunidad.

Agradecimientos

Yelina:

Agradecer en primer lugar a mí mama, no me equivoco si digo que eres la mejor del mundo y lo más importante en mi vida. Gracias por estar siempre presente, por apoyarme, por confiar en mí en todo momento y por los sacrificios que has hecho por mí. Te debo lo que soy. Te quiero mucho.

A mis abuelos, los viejitos más lindos del mundo por estar a mi lado cuando los necesito, saber cuidarme, comprenderme y malcriarme, espero se sientan orgullosos de mí. Han sido para mí abuelos, padres y amigos.

A mi tía, mi segunda madre, tengo tanto que agradecerte y sé que nunca encontraré la forma de hacerlo. Gracias por ser mi tiita linda, por tu constante apoyo y confianza, espero que sepas que ocupas un lugar muy especial.

A mi padre por ayudarme a ser la mujer que soy hoy y a mis hermanos por siempre ayudarme en lo que necesito.

A mis primos PP y Nani por cuidarme y quererme, a mis dos niñas lindas Yili y Vesy.

A mis tíos Pablin y Rubi por su apoyo y buenos consejos. A mi tío Carlito, que siempre se acuerda de mí, lo quiero mucho, nunca me olvido de él.

A mi novio que ha sabido esperarme, acompañarme y quererme como soy. En todo este tiempo nunca me he sentido sola, aunque hayamos estado lejos. A su familia por preocuparse por mí.

Agradecimientos

A Day, por siempre estar a mi lado en los momentos felices, también en los tristes. Tenemos grandes recuerdos, mi otra hermanita.
A los pocos amigos que me han acompañado, me han ayudado y se han ganado mi cariño en estos 5 años de la carrera Adita, Yayi, René, Made. He extrañado mi familia en este tiempo, y ahora los extrañare a ustedes. A Katia, Made (la flaca), Krysia, el chupo todos me han ayudado y han compartido conmigo buenos y malos momentos. A todos mis compañeros del 4102, las primeras caras que vi cuando entre a la universidad y que nunca olvidaré.

A tutora Yalice por ayudarnos con la tesis.

En general a todos los que me han ayudado a superarme.

Yelina:

A mi madre a quien le debo mi vida. Este es un logro de las dos, te quiero mucho.

A mis abuelos que han sido mi luz y mi fuerza, porque nunca han perdido la fe en mí.

A mi tiita linda, sin ella no hubiera llegado hasta aquí, te quiero mucho.

A mi chuli, por sobre todas las cosas hacerme feliz.

A mi compañero de tesis, sin su ayuda no se hubiera logrado este trabajo.

A toda mi familia, amigos, vecinos y todo el que tuve el placer de conocer en esta Universidad le dedico este trabajo.

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas hacer uso del mismo en su beneficio.

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

_____ _____

Autor: Yelina Seija Rodríguez Autor: Julio Jesús García Guevara

_____

Tutor: MSc. Yalice Gámez Batista

RESUMEN

El presente trabajo se fundamenta en la implementación de una aplicación para la simulación de procesos industriales mediante el uso de modelos matemáticos, que permitirá a los estudiantes de la especialidad Ingeniería Automática probar estrategias de control. En el curso 2008-2009 se realizó el análisis, diseño e implementación de los casos de uso críticos del Nodo Virtual de Procesos, esta es una herramienta que permite la simulación concurrente de procesos industriales en tiempo real, de modo que puedan ser insertados en lazos de control con controladores físicos o modelados. Esta aplicación tiene como principal objetivo suplir las carencias materiales que tienen las universidades en la enseñanza de la especialidad de Ingeniería Automática. Actualmente se cuenta con una primera versión del producto, pero la misma presenta no conformidades que impiden su integración a la enseñanza. Partiendo de esta propuesta inicial, en el presente Trabajo de Diploma se realizará un análisis del diseño propuesto, así como de los módulos y componentes reutilizables. Además, se desarrollará una segunda versión de la herramienta, implementando las funcionalidades necesarias para solucionar los problemas encontrados e insertándole los controladores modelados.

PALABRAS CLAVE

Nodo Virtual de Procesos, Simulación de Procesos Industriales.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Para nadie es un secreto la relevancia que tienen en la actualidad las Tecnologías de Informática y las Comunicaciones (TIC) en todos los ámbitos de la vida, ya sea en la Salud, la Economía, la Cultura y la Educación. Producto al avance que han tenido las TIC en los últimos tiempos y las nuevas potencialidades que ofrecen, se ha logrado desarrollar un área muy importante de la informática referida a la simulación de procesos.

La simulación de procesos se ha utilizado ampliamente en aspectos prácticos en muchas disciplinas. Con esta se accede a la capacidad de experimentar independientemente del sistema real. La simulación permite obviar los riesgos inherentes a la experimentación, alcanzar una completa independencia temporal y repetir el experimento un número de veces arbitrario.

En el Instituto Superior Politécnico José Antonio Echeverría (CUJAE), Facultad de Ingeniería Eléctrica, se estudia la especialidad de Ingeniería Automática. En esta carrera se imparten una serie de contenidos que proveen a los estudiantes de toda la teoría necesaria para ejercer como profesionales de la rama. Sin embargo, no siempre estos conocimientos proveen a los estudiantes de las herramientas prácticas para enfrentarse a una situación real. Esto está muy asociado a la forma tradicional en la que se imparten estos tópicos y es entonces que se hace necesario el uso de las TIC. (Gámez, 2010) Para esta especialidad se recomienda el uso de simuladores, permitiendo que un alumno pueda trabajar sobre un problema de forma gráfica y observar cómo el cambio en un determinado elemento se ve reflejado de forma inmediata en el resto, como si estuviera ante el proceso real. (Vaquero, 2007)

En la actualidad existen diversos programas que permiten realizar simulaciones de procesos industriales. Por lo general, estos realizan las simulaciones virtuales en las aplicaciones clientes, lo que limita el trabajo grupal de los estudiantes, requisito esencial para poder incluir estos programas en la docencia. Como alternativa se crea un Nodo Virtual de Procesos (NVP) que permite la simulación simultánea de diferentes procesos concurrentes sin que interfieran unos con otros. (Gámez, 2010) Las simulaciones se ejecutan en un servidor, y los usuarios tienen acceso a ellas desde clientes situados en máquinas independientes.

La Universidad de las Ciencias Informáticas (UCI), en conjunto con desarrolladores del Instituto Superior Politécnico José Antonio Echeverría (CUJAE), ha desarrollado un proyecto para la elaboración de un simulador de procesos industriales. Esta aplicación, también conocida como Nodo Virtual de Procesos (NVP) debido al uso que hace de los mismos en su funcionamiento, está dirigida a los estudiantes de la especialidad Ingeniería Automática de la CUJAE. (Gámez, 2010)

La aplicación implementada está dividida en dos partes siguiendo la filosofía del NVP: un servidor donde se desarrolla la simulación de los procesos y clientes donde trabajan los usuarios. Tanto el cliente como el servidor son aplicaciones de escritorio que utilizan el framework Qt y C++ como lenguaje de programación. Se utilizó como estándar de acceso a la base de datos, ODBC. Para reducir el consumo de recursos en las máquinas clientes se desarrolló una variante del cliente web en PHP 5.0, usando ExtJs como framework para el diseño de la capa de presentación, como entorno de desarrollo NetBeans 6.8 y como servidor Web Apache.

El equipo de desarrollo realizó un análisis de las funcionalidades que hasta el momento presenta el NVP, de los requisitos que este debe cumplir y de las recomendaciones señaladas en trabajos de diploma (Escobar, 2008), (Gonce, 2008), (Cleger, 2009), (Hernández, 2009), (Rivero, 2009), (Muñoz, 2010), que han abordado y sustentado el proceso de desarrollo del producto.

Después del análisis de los trabajos de diplomas antes mencionado se llegó a la conclusión que la versión actual del NVP presenta varias no conformidades.

En el acoplamiento de las dos partes se presentaron serios problemas en el funcionamiento de la herramienta implementada debido al no respeto del diseño propuesto, por parte de los diplomantes de la CUJAE. Entre las no conformidades se pueden mencionar que se producen conflictos en el acceso a la base de datos, la implementación de los hilos no permite la simulación concurrente, problemas para limitar el número de usuarios conectados al sistema y por procesos, problemas con el envío de datos desde el servidor a los clientes. Además, quedaron pendientes de implementar el sistema de reportes, garantizar que se simule el sistema en tiempo real y la incorporación de controladores modelados para su simulación.

Atendiendo a la situación problemática planteada anteriormente deviene el siguiente problema: El NVP no cumple con los requerimientos definidos, dificultando la incorporación de la herramienta al proceso docente de los estudiantes de la carrera de Ingeniería Automática.

Objeto de estudio: Proceso de desarrollo de software.

Campo de acción: Rediseño e implementación de una herramienta interactiva para la simulación de procesos industriales: Nodo Virtual de Procesos.

Objetivo general: Desarrollar una solución informática que elimine las no conformidades detectadas para lograr la incorporación del NVP al proceso docente de los estudiantes de la especialidad de Ingeniería Automática.

Idea a defender: Desarrollando una herramienta interactiva para la simulación de procesos industriales que elimine las no conformidades detectadas por el equipo de desarrollo, se lograría la incorporación del NVP al proceso docente de los estudiantes de la especialidad de Ingeniería Automática.

Del objetivo general se derivan los siguientes **objetivos específicos** y **tareas**:

- *Identificar las no conformidades.*

Tarea 1: Identificación de los problemas del diseño y del funcionamiento del NVP actual.

Tarea 2: Identificación de los casos de uso por implementar.

- *Elaborar el marco teórico.*

Tarea 3: Revisión y análisis de las bibliografías correspondientes al tema.

- *Analizar los recursos y vías de solución que resuelvan las no conformidades.*

Tarea 4: Revisión de las técnicas, tecnologías y metodologías que den solución de forma óptima a las no conformidades detectadas.

Tarea 5: Análisis de las posibles soluciones a las no conformidades detectadas.

- *Realizar análisis y diseño de la herramienta.*

Tarea 6: Rediseño de la aplicación, que permita aprovechar lo implementado y que solucione las no conformidades detectadas.

Tarea 7: Análisis y diseño del módulo de inserción en el lazo de control de controladores tanto físicos como modelados.

- *Implementar la nueva versión del NVP.*
- *Validar la solución implementada.*

Tarea 8: Desarrollo de los casos de prueba que permitan validar la solución.

Métodos de Investigación:

Métodos Teóricos

Inductivo-Deductivo: Se hace uso de deducciones para llegar a tener una visión clara de lo que se quiere hacer y así adquirir nuevos conocimientos. Este método se aplica en inducción y deducción de los lenguajes de programación que se van a escoger.

Hipotético-deductivo: A partir de la interpretación de la realidad se establecen posibles situaciones o resultados para llegar a conclusiones.

Histórico-Lógico: Se realiza un análisis de la evolución de las diferentes herramientas de simulación que hacen uso de nodos virtuales, obteniendo una tendencia de cómo se debe comportar en la actualidad.

Métodos Empíricos:

La observación: Mediante guías de observación se le dará seguimiento al desarrollo de la aplicación.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 INTRODUCCIÓN

En este capítulo se exponen las diferentes herramientas que utilizan los nodos virtuales para determinar las características y funcionalidades que deben cumplir los mismos. Se realiza un análisis a las posibles vías de solución que resuelvan las no conformidades detectadas. Además, un estudio de las tecnologías actuales y de las principales herramientas que pudieran ser adecuadas para la construcción del sistema que se pretende desarrollar.

1.1.1 NODO VIRTUAL DE PROCESOS

En la actualidad existen muchas herramientas informáticas que permiten realizar simulaciones de procesos industriales (en tiempo real o no), pero todas tienen limitaciones en cuanto al número de recursos que necesitan para su implementación o en cuanto al número de procesos simultáneos que se pueden simular. Incluso, en su mayoría, o simulan los procesos o permiten probar aplicaciones reales, nunca las dos prestaciones. (Gámez, 2010)

Para dar solución a esta problemática se optó por un Nodo Virtual, el cual es una aplicación que permite ejecutar diferentes instancias del software en un único nodo (nodo físico) y cada instancia del software trabaja en un entorno de ejecución independiente (nodo virtual). (Maier, 2005)

Atendiendo a esto se define como nodo virtual de procesos, el software que permitirá implementar los modelos de diferentes procesos para su simulación o para probar aplicaciones en tiempo real. Será necesario que varios procesos se encuentren activos de manera simultánea, lo que requiere de gran cantidad de nodos y computadoras que no están disponibles generalmente. (Gámez, 2010)

Para ello se establece como nodo a aquella estructura a la cual se interconectan varios elementos. No hay que pensar en un nodo como un elemento constituido solamente por una parte física, sino más bien considerarlo como una unidad funcional en donde tiene que haber tanto hardware como software. (Hernández, 2009)

Por otra parte, al ser el punto de conexión de dos o más elementos, el nodo por lo general tiene la capacidad de recibir información, procesarla y enrutarla a uno o varios nodos. De esta forma, un nodo puede ser el punto de conexión para transmitir los datos, el punto desde el cual se distribuyan los datos hacia otros nodos y el punto al que se transmitan los datos. (Escobar, 2008) Esta filosofía de trabajo permite la simulación simultánea de diferentes procesos concurrentes sin que interfieran unos con otros. La virtualización de nodos provee una vía de regular el acceso a recursos de hardware exclusivos de un determinado número de consumidores. En este caso los consumidores son los entornos de ejecución para cada proceso, los cuales están sujetos a las propiedades de la simulación.

De ahí se derivan los siguientes requerimientos para la virtualización del nodo: (Gámez, 2010)

- El parámetro más importante es minimizar los gastos de virtualización para preservar los recursos para el proceso en ejecución.

- Cada entorno de ejecución introducido por la virtualización del nodo debe ser tan transparente como sea posible para los restantes. Esto es importante para que la medición de la implementación no sufra modificaciones en comparación con la real.

En el mundo se han desarrollado varias aplicaciones que utilizan los nodos virtuales, con el objetivo de aprovechar al máximo los recursos de las computadoras. A continuación se mencionan algunos programas que hacen uso de los nodos virtuales.

En la Universidad de Nottingham Trent, se creó un software para la simulación de sistemas de agua (Hosseinzaman, 1995) surgido por la necesidad que tenían en la industria del agua de adquirir y almacenar datos de estaciones remotas para la inspección ingenieril. Esta aplicación no es la solución pues no está concebida para simular procesos independientes, sino que simula subprocesos de un sistema específico, para luego integrarlo en el proceso general como un todo. (Gámez, 2010)

En la Universidad de Stanford se creó un algoritmo de nodos virtuales para el trabajo con imágenes en tres dimensiones. (Molino, 2004) Un elemento es fragmentado para crear varias réplicas del elemento y se le asigna una porción real del material a cada réplica. De esto resultan elementos que contienen material

real y regiones vacías. El algoritmo del nodo virtual determina automáticamente el número de réplicas así como la asignación de material para cada una. Este simulador tampoco satisface las necesidades planteadas.

En la Universidad de Stuttgart en Alemania, instituto especializado en sistemas distribuidos y paralelos se creó “Network Emulation Testbed” para la simulación de redes. (Maier, 2005) La aplicación simula un entorno de redes configurables que reproduce un escenario real de cientos de nodos en comunicación. Esto permite medir de manera comparativa el comportamiento de una aplicación en diferentes entornos de redes o de varias aplicaciones en un mismo entorno de red. La principal limitación de esta aplicación está en que es diseñada para simular redes, por lo que no es posible utilizarla para la simulación de procesos que tienen una dinámica más compleja y variada. (Gámez, 2010)

En Japón se creó la herramienta “StarBED” para dar solución al vacío que existe entre Internet y los entornos para experimentación, atendiendo a los aspectos de escala, complejidad y realidad. (Miyachi, 2006) Es una aplicación de pruebas basada en lotes de nodos que tiene como objetivo construir entornos de experimentaciones reales, complejos y de gran escala.

Por todo lo antes expuesto se concluye que a pesar de que existen varias aplicaciones que hacen uso de los nodos virtuales, no se ha identificado una herramienta informática que funcione como un nodo virtual de procesos, y específicamente, que simule procesos industriales.

La ausencia de un software con estas características motivó el desarrollo del simulador de procesos industriales por la UCI en conjunto con la CUJAE, aunque la versión implementada generó no conformidades que hicieron imposible su integración en el proceso de enseñanza – aprendizaje de la especialidad Ingeniería Automática.

1.2 Análisis de posibles soluciones para las no conformidades

Como se abordó en la introducción, la versión existente del NVP presenta numerosas no conformidades que han hecho imposible su integración al proceso de enseñanza – aprendizaje de la especialidad

Ingeniería Automática. A continuación se enumeran las principales no conformidades encontradas y se hace un análisis de las posibles soluciones, para seleccionar las más adecuadas al problema.

1.2.1 HILOS DE EJECUCIÓN EN LA APLICACIÓN

La filosofía del nodo virtual de procesos se fundamenta principalmente en la simulación paralela de varios procesos concurrentes sin que intervengan unos con otros. Para lograr la simultaneidad en la ejecución de múltiples tareas interactivas en la versión implementada del NVP se hizo uso de los hilos de ejecución. Los hilos de ejecución básicamente permiten que una tarea sea ejecutada de forma paralela con otra tarea en entornos independientes.

La aplicación implementada del NVP no permite la simulación concurrente de varios procesos, por tanto, se hace necesario analizar qué es lo que no funciona correctamente en la implementación de los hilos y buscar las posibles vías de solución.

Dos vías de solución: hilos y procesos

En el estudio realizado se encontró que para lograr que determinadas tareas se ejecuten de manera concurrente pueden ser por un conjunto de procesos o por el uso de los hilos de ejecución. Entre los procesos y los hilos existen diferencias y limitaciones. Los hilos se distinguen de los tradicionales procesos en que los procesos son independientes, llevan bastante información de estados e interactúan sólo a través de mecanismos de comunicación dados por el sistema. Por otra parte, muchos hilos generalmente comparten otros recursos de forma directa. Los hilos dan la facilidad de ser más rápidos a la hora de cambiar de un hilo a otro dentro del mismo proceso, que cambiar de un proceso a otro. Este fenómeno se debe a que los hilos comparten datos y espacios de direcciones, mientras que los procesos, al ser independientes, no lo hacen. Al cambiar de un proceso a otro se genera lo que se conoce como tiempo desperdiciado por el procesador para realizar un cambio de contexto, en este caso pasar del estado de ejecución al estado de espera y colocar el nuevo proceso en ejecución. En los hilos, como pertenecen a un mismo proceso, al realizar un cambio de hilo el tiempo perdido es casi despreciable.

Ventajas de los hilos

- Los hilos son generados a partir de la creación de un proceso, se puede decir que un proceso es un hilo de ejecución, por tanto, los beneficios de los hilos se derivan de las implicaciones de rendimiento.
- Se tarda mucho menos tiempo en crear un hilo nuevo en un proceso existente que en crear un proceso.
- Se tarda mucho menos tiempo en cambiar entre dos hilos de un mismo proceso.
- Los hilos aumentan la eficiencia de la comunicación entre programas en ejecución.
- Al terminar un proceso, los hilos asociados a él también terminan.

Implementación de los hilos en la versión anterior

En la versión anterior del NVP se hizo uso de los hilos como muestra la figura 1.1. Se creó un hilo principal que se encarga de actualizar a los usuarios cada vez que se tiene un dato nuevo como resultado de la simulación. Por otra parte, cada vez que un proceso es activado para realizar su simulación se crea un nuevo hilo, el cual se ejecuta sin interferir en la simulación de cualquier otro proceso. (Gámez, 2010)

La comunicación entre los hilos encargados de la simulación de los procesos y el hilo principal se efectúa mediante Signals y Slots.

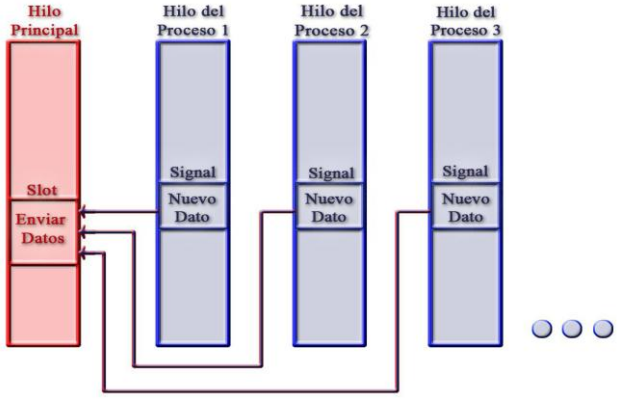


Figura 1.1 Esquema de hilos utilizados en la versión anterior.

Teniendo en cuenta las ventajas que ofrece el uso de los hilos de ejecución se decidió que en la versión a desarrollar se mantendrá el uso de los mismos. No obstante, en el NVP cada proceso va a tener asociado un hilo de ejecución y cada cliente una vez que solicite simular un proceso ya sea el proceso inicializado por él o un proceso ya inicializado va a tener un socket, por lo que se concluye que para cada proceso que se esté simulando va a existir un hilo de ejecución y para cada cliente va a existir un socket.

1.2.2 DISEÑO Y FUNCIONALIDADES DE LA APLICACIÓN

En la herramienta con la que se cuenta quedaron pendientes de implementación el sistema de reportes, garantizar que se simule el sistema en tiempo real, limitar el número de usuarios conectados al sistema y por procesos. Además, de la existencia de conflictos en la base de datos y que debe incorporarse al diseño del módulo de lazos de control de controladores físicos y modelados.

Con el objetivo de eliminar todas estas no conformidades se realizará primeramente un refinamiento de los requisitos, la reestructuración del modelo del sistema, así como el análisis y el diseño, para luego pasar a la fase de implementación. En esta última fase se decidió usar un estándar de codificación apropiado con el propósito de que el código sea más organizado y legible, listo para próximas mejoras y mantenimiento.

La simulación en tiempo real se garantizará haciendo uso de los sockets definidos como usuarios, donde cada usuario va a tener un socket asociado y los procesos definidos como hilos. El sistema vinculará los sockets con los hilos, permitiendo que cada uno acceda al nuevo valor generado simultáneamente, de tal forma que se asegure la obtención de las variables en tiempo real.

Para implementar el sistema de reportes se almacenará en la base de datos todas las acciones realizadas por el usuario, así como la fecha y la hora. Esta tabla en conjunto con las otras va a generar la información suficiente para definir los reportes necesarios para la gestión de la aplicación y aquellos que sean importantes al administrador.

Para proveer un entorno de procesos configurables basado en estándares, que permita a cada miembro del equipo un fácil acceso a una base de conocimiento con guías, plantillas y herramientas para todas las

actividades, el equipo de desarrollo debe regirse por una metodología de desarrollo de software. Así mismo, para garantizar que esta base de conocimientos se obtenga en tiempo y forma debe utilizarse alguna herramienta que soporte el ciclo completo de desarrollo de software de acuerdo con la metodología seleccionada.

Teniendo en cuenta las características requeridas en el NVP y la necesidad de desarrollar una herramienta en software libre, se procede a realizar un estudio comparativo entre las metodologías de diseño de software y entre las plataformas de programación para seleccionar las más acordes.

1.3 METODOLOGÍAS DE DESARROLLO DE SOFTWARE

En el contexto mundial actual existen tendencias en el desarrollo de software que se van imponiendo para lograr productos de alta calidad y bajos costos de elaboración. La reusabilidad de código, extensibilidad del producto y la genericidad de las aplicaciones son algunos ejemplos de estas tendencias. Se pueden citar otras como el usar varios sistemas gestores de base de datos o programar aplicaciones que sean independientes de la plataforma de almacenamiento de datos, el uso de una metodología de desarrollo de software como el Proceso Unificado Racional (*RUP*) o la Programación Extrema (*XP*). (Cuevas, 2007)

El término Metodología se define como un conjunto de métodos eficientes orientados a conseguir un objetivo propuesto. Son un conjunto de procesos que organizados dan una secuencia de pasos a seguir para obtener los hitos propuestos y finalmente el producto final. (López, 2005)

Metodologías como las antes mencionadas constituyen un factor importante para lograr productos con calidad en el tiempo requerido. Actualmente existen múltiples metodologías, todas con muchas características comunes, pero que comparten también significativas diferencias, por lo que se recomienda hacer un estudio para realizar su correcta selección.

1.3.1 PROGRAMACIÓN EXTREMA

La Programación Extrema (XP), es una de las metodologías de desarrollo de software con más éxito en la actualidad. Se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. Se utiliza en proyectos con equipos de desarrollo pequeños y con plazos de entrega cortos.

La metodología consiste en una programación rápida o extrema. Una particularidad es tener como miembro del equipo al usuario final. (Welicki, León E)

Lo fundamental de XP es: (López, 2005)

La comunicación: Entre los usuarios y los desarrolladores.

La simplicidad: Al desarrollar y codificar los módulos del sistema.

La retroalimentación: Concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.

Esta metodología ha sido diseñada para solucionar el eterno problema del desarrollo de software por encargo: entregar el resultado que el cliente necesita a tiempo. La Programación Extrema no es la metodología de desarrollo de software adecuada para utilizar en la creación del NVP puesto que esta concibe el flujo de trabajo de requisitos con un excesivo dinamismo sin hacer hincapié en el seguimiento al cambio de los mismos.

1.3.2 SCRUM

El Scrum es un proceso de desarrollo iterativo e incremental enfocado a la gestión de procesos de desarrollo de software. En Scrum se realizan entregas parciales y regulares del resultado final del proyecto, priorizadas por el beneficio que aportan al receptor del mismo. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, los requisitos son cambiantes o poco definidos. La innovación**,** la competitividad y la productividad son fundamentales. (Xavier, 2009)

Scrum se focaliza en priorizar el trabajo en función del valor que tenga para el negocio, maximizando la utilidad de lo que se construye y el retorno de inversión. Está diseñado especialmente para adaptarse a los cambios en los requerimientos, los cuales son revisados y ajustados durante el proyecto en intervalos muy cortos y regulares. De esta forma, se puede adaptar en tiempo real el producto que se está construyendo a las necesidades del cliente, dicho de otra manera, incorporar cambios con rapidez y en cualquier fase del proyecto. (Palacio, 2006)

Scrum como metodología de desarrollo de software no es recomendable para algunos contextos y proyectos de software, donde el cliente no pueda estar en colaboración permanente con el equipo, condiciones que presenta el proyecto de creación del NVP, por lo que su aplicación al mismo no es recomendable.

1.3.3 Proceso Unificado de Desarrollo

RUP es un proceso de desarrollo de software, definido como un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software. (Jacobson, 2000) Sin embargo, el proceso unificado es más que un proceso de trabajo, es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones y diferentes niveles de aptitud. (Gámez, 2010)

Características de RUP: (Jacobson, 2000)

Dirigido por casos de uso, especifican las funcionalidades que el sistema le proporciona al usuario.

Centrado en la arquitectura, es la manera en que se organiza el sistema, depende de los casos de uso claves y debe tener en cuenta la comprensibilidad, la facilidad de adaptación al cambio y la reutilización. Los casos de uso claves son aquellos que dotan al sistema con la funcionalidad fundamental para los usuarios y sin los cuales el resto de los casos de uso no tienen sentido.

Iterativo e incremental, en todo el proceso de desarrollo se producen versiones incrementales (que se acercan al producto terminado) del producto en desarrollo.

Basado en componentes y utiliza **UML** (Lenguaje de Modelado Unificado) para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software.

Después de realizar el estudio de las principales metodologías de desarrollo de software se tomó RUP como proceso rector por adaptarse a las características y complejidad de este proyecto de software. Como una plataforma de procesos que abarca todas las prácticas de la industria, configurable para

proyectos pequeños, permitiendo seleccionar fácilmente el conjunto de componentes de proceso que se ajustan a las necesidades específicas del proyecto.

RUP constituye la metodología de desarrollo más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. El NVP no es un sistema de compleja construcción, pero requiere que su realización sea a largo plazo y con un equipo de desarrollo numeroso. Una particularidad de esta metodología es que, en cada ciclo de iteración se hace exigente el uso de artefactos, siendo por este motivo una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software. Durante el proceso de desarrollo del NVP no se va a interactuar con el cliente por lo que dichos artefactos formarán parte del producto final y serán la clave para obtener un software robusto.

1.4 Lenguaje de Modelación Unificado (UML)

UML (Unified Modeling Language), como sus siglas indican en español Lenguaje de Modelación Unificado es un lenguaje gráfico para detallar, construir, visualizar y documentar las partes o artefactos (información que se utiliza o produce mediante un proceso de software). Pueden ser artefactos: un modelo, una descripción que comprende el desarrollo de software que se base n en el enfoque Orientado a Objetos.

UML es un lenguaje más expresivo, claro y uniforme que los anteriores definidos para el diseño Orientado a Objetos, que no garantiza el éxito de los proyectos pero si mejora sustancialmente el desarrollo de los mismos, al permitir una nueva y fuerte integración entre las herramientas, los procesos y los dominios (Pressman, 2005).

Como lenguaje de modelado se utilizará UML debido a que la metodología de desarrollo antes seleccionada, RUP, está soportada por este lenguaje.

1.5 Herramientas de modelado

En el mundo informático se han creado diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste en términos de tiempo y de dinero. Estas son denominadas herramientas CASE (Computer Aided Software Engineering), Ingeniería de Software

Asistida por Ordenador y brindan un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación pasando por el análisis y diseño, hasta la generación del código fuente de los programas y la documentación. Las herramientas CASE tienen como objetivo primordial:

- Mejorar la productividad en el desarrollo y mantenimiento del software.
- Aumentar la calidad del software.
- Reducir el tiempo y coste de desarrollo y mantenimiento de los sistemas informáticos.
- Mejorar la planificación de un proyecto.
- Aumentar la biblioteca de conocimientos informáticos de una empresa ayudando a la búsqueda de soluciones para los requisitos.
- Automatizar el desarrollo del software, la documentación, la generación de código, las pruebas de errores y la gestión del proyecto.
- Ayuda a la reutilización del software, portabilidad y estandarización de la documentación.
- Gestión global en todas las fases de desarrollo de software con una misma herramienta.
- Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

Entre las herramientas CASE orientadas a UML se pueden encontrar: Rational Rose y Visual Paradigm.

1.5.1 RATIONAL ROSE

Es una de las más poderosas del modelado visual para el análisis y diseño de sistemas basados en objetos. Se utiliza para modelar un sistema antes de proceder a construirlo. Cubre todo el ciclo de vida de un proyecto.

Características:

- Soporte para análisis de patrones ANSI C++, Rose J y Visual C++ basado en "Design Patterns: Elements of Reusable Object-Oriented Software".
- Característica de control por separado de componentes modelo que permite una administración más granular y el uso de modelos.
- Generación de código ANSI C ++, C++, CORBA, Java y Visual Basic, con capacidad de sincronización modelo- código configurable.
- El Add-In para modelado Web provee visualización, modelado y las herramientas para desarrollar aplicaciones Web.
- Capacidad de crear definiciones de tipo de documento XML (DTD) para el uso en la aplicación.
- Publicación web y generación de informes para optimizar la comunicación dentro del equipo.

1.5.2 VISUAL PARADIGM

Visual Paradigm para UML es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado ayuda a una construcción más rápida de aplicaciones de calidad, mejores y a un menor coste, además de permitir el dibujo de todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación, así como una serie de tutoriales con demostraciones interactivas y proyectos.

Características generales: (SlideShare, 2009)

- Soporte de UML versión 2.1.
- Diagramas de Procesos de Negocio: Proceso, Decisión, Actor de negocio, Documento.

- Interoperabilidad con modelos UML2 (meta modelos UML 2.x para plataforma Eclipse) a través de XMI.
- Ingeniería inversa: Código a modelo, código a diagrama, ingeniería inversa Java, C++, Esquemas XML, XML, .NET exe/dll, CORBA IDL.
- Generación de código: Modelo a código, diagrama a código.
- Editor de Detalles de Casos de Uso: Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Generación de base de datos: Transformación de diagramas de Entidad-Relación en modelos de base de datos.
- Ingeniería inversa de base de datos: Desde sistemas gestores de base de datos (SGBD) existentes a diagramas de entidad-relación.
- Generador de informes para generación de documentación.
- Distribución automática de diagramas: Reorganización de las figuras y conectores de los diagramas UML. Importación y exportación de ficheros XMI.

Partiendo del estudio realizado a las principales herramientas CASE, se decidió utilizar Visual Paradigm. Se hace esta selección principalmente por ser una herramienta multiplataforma, facilidad que pocas herramientas CASE brindan. Permite exportar documentos, es robusta y de fácil uso. Además, se sustenta la elección de la misma en el hecho de que la UCI cuenta con la licencia para su uso.

1.7 Lenguaje de Programación y Entorno de Desarrollo

Para la construcción de este proyecto se compararon numerosas alternativas de desarrollo. Desde usar un único lenguaje de alta eficiencia como C++ y basarse en librerías intermedias para lograr portabilidad,

hasta lenguajes que no dependieran en absoluto de la máquina de ejecución como puede ser Java. Pero sin perder de vista que para el objetivo fundamental de las operaciones también es necesario altas capacidades de cálculo en punto flotante y una eficiente gestión de la memoria. (Rodríguez, 2005)
Dentro de los lenguajes de programación que se consideraron están Java, C++ y C#.

1.7.1 LENGUAJE C#

C# es un lenguaje orientado a objetos con seguridad de tipos que permite a los desarrolladores crear una amplia gama de aplicaciones sólidas y seguras que se ejecutan en .NET Framework. Se puede utilizar para crear aplicaciones cliente para Windows tradicionales, servicios Web XML, componentes distribuidos, aplicaciones cliente-servidor, aplicaciones de base de datos, y muchas tareas más. (Garrigó, 2009)

Es una versión avanzada de C y C++, diseñado especialmente para el entorno .NET. Amplía las capacidades de otros lenguajes proporcionando un entorno de desarrollo completo para crear aplicaciones. C# mezcla la potencia de C, las capacidades de orientación a objetos de C++ y la interfaz gráfica de Visual Basic. Además, los programas en C# y en Java se compilan a bytecode[1].

1.7.2 LENGUAJE C++

Versión de C orientada a objetos creada por Bjarne Stroustrup. C++. Se ha popularizado porque combina la programación tradicional en C con programación orientada a objetos. Las principales características del C++ son abstracción (encapsulación), el soporte para programación orientada a objetos (polimorfismo) y el soporte de plantillas o programación genérica (Templates). Desde sus inicios, intentó ser un lenguaje que incluyera completamente al lenguaje C (quizá el 99% del código escrito en C es válido en C++) pero al mismo tiempo incorpora muchas características sofisticadas, tales como: Programación orientada a Objetos, excepciones, sobrecarga de operadores, templates o plantillas. (Wesley, 2001)

[1] Son códigos de bytes no asociados a una plataforma y que son generados por los compiladores de Java y C# y que pueden ser transferidos a cualquier plataforma.

1.7.3 Lenguaje Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principio de los años 90. Es moderno, de alto nivel y recoge los elementos de programación que típicamente se encuentran en todos los lenguajes, permitiendo la realización de programas profesionales. (Jesús, 2005)

La sintaxis del lenguaje heredó características de C y C++, explícitamente eliminando aquellas que para muchos programadores (según los diseñadores) resultan excesivamente complejas e inseguras.

1.7.4 Criterios de Comparación

Comparar lenguajes de programación nunca ha sido una tarea sencilla ni objetiva. Teniendo en cuenta los requerimientos del software se establecieron los siguientes criterios de comparación:

1. Portabilidad.
2. Capacidades 2D/3D.
3. Matemática de precisión compleja.
4. Gestión de memoria.
5. Velocidad de ejecución.
6. Licencia.
7. Eficiencia
8. Modularidad.

1.7.5 Elección del lenguaje a utilizar

Después de un estudio a los lenguajes de programación, se realizó una tabla resumen (Tabla 1.1) donde se le asignaron valores a los diferentes criterios de selección en función de su importancia para el desarrollo de la aplicación. A continuación se muestran los resultados:

Tabla 1.1 Comparación entre los lenguajes de programación.

Criterio de selección	Peso	C++	C#	Java
Portabilidad	5	3	4	5

Capacidad 2D\3D	5	3	2	2
Matemática de precisión	5	3	2	2
Gestión memoria	5	5	3	3
Velocidad	10	10	8	6
Licencia	10	10	0	10
Eficiencia	10	10	6	6
Modularidad	5	4	4	4
Total		**48**	**29**	**38**

Teniendo en cuenta estos resultados se optó por escoger como lenguaje de programación C++ para aprovechar su velocidad de ejecución, eficiencia y todas las potencialidades que ofrece de manera general.

1.7.6 ELECCIÓN DEL ENTORNO DE DESARROLLO

Para la elección de entorno de desarrollo integrado (IDE) se tuvo en cuenta que fuera una aplicación sobre software libre por las ventajas que conlleva. Entre estas se destacan:

- Evita la dependencia tecnológica de empresas foráneas.
- Ahorros por pagos de licencias de software.
- Posibilidad de revisar el código fuente.

Entre los IDE que existen para Linux los más populares son:

KDevelop: Surgió en 1998 con el fin de desarrollar un IDE fácil de usar para KDE (K Desktop Environment). Desde entonces está públicamente disponible bajo licencia GPL y soporta lenguajes de programación como: C, C++, Java, SQL, Python, Perl y Pascal. Sólo corre en sistemas Linux y otros sistemas Unix. Su última versión es la 3.5 y salió el 16 de octubre del 2007. Tiene como limitantes principales que su entorno gráfico es muy pobre y sólo corre sobre plataforma Linux. (www.kdevelop.org)

Eclipse: Es un IDE multiplataforma desarrollado por IBM. En la actualidad lo mantiene la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos, capacidades y servicios complementarios. Pese a que está escrito en su mayor parte en Java (salvo el núcleo), se ejecute sobre la máquina virtual y su uso más popular sea como un IDE para Java; Eclipse es neutral y adaptable a cualquier tipo de lenguaje, por ejemplo C/C++, Cobol, C#, XML, etc. Sus mayores ventajas radican en su gran comunidad de desarrollo que lo ubica como el mejor IDE Java. (www.eclipse.org)

Qt: Framework C++ para Desarrollo Multiplataforma*:* La librería Qt es desarrollada por la empresa noruega TrollTech, con licencias tanto libre como comercial, desarrollada en C++ con añadidos. Qt a la fecha en sus versiones 4.X es un toolkit maduro que cuenta aproximadamente con 500 clases, más de 9000 funciones y 500.000 líneas de código. Le da al programador mucha de la potencia que le brindan lenguajes como C# o Java con la eficiencia de código compilado en C++.

Qt, al igual que otros toolkits no sólo cuenta con clases para la construcción de interfaces de usuario, también incluyen soporte para dibujo en 2D, hilos, red y base de datos. La diferencia fundamental entre Qt y las otras librerías es que Qt le agrega al C++ estándar los conceptos de *signal* y *slot*, los cuales son similares en funcionalidad al concepto de callback.[2] (qt.nokia.com)

[2] Devolución de llamada o retrollamada en español: Es una llamada es una función que recibe como argumento la dirección o puntero de otra función, cuando la retrollamada es llamada esta recurre al puntero de la función y la ejecuta.

Se seleccionó Qt debido a que simplifica el desarrollo de aplicaciones desktop para programadores C++. El marco ofrece amplia funcionalidad y herramientas intuitivas para el desarrollo de aplicaciones avanzadas. Le permite a los desarrolladores crear potentes aplicaciones de subprocesos múltiples que funcionan en todos los sistemas operativos principales desde una única base de código fuente. Las características potentes de este framework incluyen funciones para la administración de subprocesos, objetos y datos, como también mecanismos directos para los subprocesos de comunicación interna, facilitan y aceleran la programación paralela.

1.8 Selección del estándar de conexión de la Base de datos

La conexión y el intercambio de información entre las aplicaciones y los gestores de base de datos (SGBD) han tenido su máximo exponente con el surgimiento del lenguaje de consulta estructurado (SQL). Gracias a este y al surgimiento de otras condiciones favorables, como su adopción por parte de un gran número de fabricantes de SGBD y a un mutuo consenso de estos en el diseño de los SGBD ha sido posible la creación y desarrollo de varios estándares para establecer dicha conexión.

A continuación se hace un breve estudio de los principales estándares y bibliotecas de acceso a datos:

Object Linking and Embedding for Databases (Enlace e incrustación de objetos para base de datos u OLEDB) es una tecnología desarrollada por Microsoft para tener acceso a diferentes base de datos de manera uniforme. Permite separar los datos de la aplicación que los requiere. Está dividido en consumidores y proveedores; el consumidor es la aplicación que requiere acceso a los datos y el proveedor es el componente de software que expone una interfaz OLEDB a través del uso del Component Object Model (COM) (OpenLink, 2004)

Open Database Connectivity (Conectividad abierta a base de datos u ODBC) es un estándar de acceso a base de datos desarrollado por Microsoft Corporation, aunque se han desarrollado diferentes implementaciones del mismo, tanto libres como propietarias. Por otra parte, los fabricantes de un gran número de gestores de base de datos, como Access, PostgreSQL, MySQL, Oracle y Microsoft SQL Server, han implementado drivers para los mismos compatibles con este estándar. Permite acceder a cualquier base de datos desde cualquier aplicación sin importar que sistema gestor de base de datos

(SGBD) se utilice. Inserta una capa intermedia entre la aplicación y el SGBD, cuyo propósito es traducir las consultas de datos de la aplicación en comandos que el SGBD entienda. Para que esto funcione tanto la aplicación como el SGBD deben ser compatibles con ODBC, esto es que la aplicación debe ser capaz de producir comandos ODBC y el SGBD debe ser capaz de responder a ellos. Desde la versión 2.0 el estándar soporta SAG y SQL. (Schultz, 2000) (Ripley, 2010)

Atendiendo a las anteriores características señaladas de cada estándar, se decidió utilizar ODBC para establecer la conexión a la base de datos. Por otra parte, es necesario destacar que el framework Qt utilizado para desarrollar el NVP, es compatible con dicho estándar.

1.9 CONCLUSIONES PARCIALES

En el presente capítulo se realizó un análisis de las diferentes soluciones posibles para resolver las no conformidades presentadas por la versión anterior del NVP, escogiéndose las más favorables a la situación planteada.

La simulación concurrente se logrará con el uso de los hilos de ejecución y los sockets. Para la implementación del sistema de reportes se contará con una tabla en la base de datos donde estarán registradas las acciones del usuario, que el sistema consultará para generar los reportes.

Se realizó una comparación entre las metodologías de diseño de software más usadas y teniendo en cuenta las características de la aplicación se seleccionó la metodología RUP como la más idónea. En base a esta decisión se define como lenguaje de modelo el UML, y por las ventajas que ofrece se utilizará el Visual Paradigm como herramienta de modelado.

De igual forma se hizo un estudio de los diferentes lenguajes de programación y se seleccionó el C++ por todas las ventajas que ofrece en cuanto a velocidad y eficiencia entre otros parámetros. Se escogió el framework Qt para la implementación del nodo virtual de procesos. Además se optó por usar el estándar de conexión ODBC.

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

2.1 INTRODUCCIÓN

En este capítulo se hace una descripción de las distintas soluciones que se le dan a los problemas descritos y las ventajas que implican sobre el diseño con que anteriormente se contaba. De forma general se le da solución a los problemas existentes en el NVP, se modifica el diseño de algunas partes del sistema para dar cabida a funcionalidades que no eran soportadas con el diseño que anteriormente se contaba. Se reestructura la documentación y los datos.

2.2 REFINAMIENTO DE REQUISITOS

En el curso 2008-2009 se presentó el Trabajo de Diploma *Análisis y Diseño de una herramienta interactiva de simulación de procesos: Nodo Virtual de Procesos Segunda Iteración,* en el cual se realizó un refinamiento de requisitos para la construcción de la herramienta. En el presente trabajo se revisaron dichos requisitos debido al surgimiento de ambigüedades y la inadecuada administración de los mismos. Dentro de la mala administración se encuentran los factores como requerimientos incompletos y mal manejo de los cambios. Además de que muchos de los requisitos especificados no se contemplan entre las funcionalidades de la aplicación implementada, ya sea de forma parcial o completa.

Para llevar a cabo el proceso de revisión se realizaron algunos de los pasos contenidos en las actividades de la Ingeniería de Requisitos. Se comenzó por la Elicitación debido a la necesidad de volver a estudiar la documentación referente al NVP. Se realizaron entrevistas con la parte interesada con el fin de captar información lo más exacta, clara y abarcadora posible, de forma tal que se lograra un refinamiento de los requisitos capturados hasta el momento. Durante la Especificación se describieron las restricciones, funcionalidades y características que tendrá el sistema a desarrollar. Además se identificaron los datos que entrarán y saldrán del sistema. La calidad del refinamiento realizado en las actividades anteriores se evaluó mediante la Validación, revisándose las especificaciones para asegurarse de que los requisitos no son ambiguos, inconsistentes y que los errores detectados fueron corregidos. A continuación se registran los requisitos capturados.

2.2.1 REQUISITOS FUNCIONALES

Los requerimientos funcionales de un sistema describen las funcionalidades o los servicios que se espera que ofrezca a los usuarios finales. Estos dependen del tipo de software, del sistema que se desarrolle y de los posibles usuarios del software.

A continuación se exponen los requisitos funcionales obtenidos a partir de la extracción de la información de los procesos que se desean automatizar en el NVP. Se detalla por separado los requisitos que en la versión anterior se contemplan entre sus funcionalidades, los requisitos que no, los que se cumplieron parcialmente y los requisitos que refieren a nuevas funcionalidades.

Los requisitos funcionales del sistema están expresados en lenguaje natural. Los mismos serán identificados con las siglas **RF** más el **Número del requisito** (Ej. RF1) y el requisito. Las funciones insertar, eliminar, modificar y mostrar listado están englobados en el requisito gestionar.

RF1: Conectar al sistema.
RF2: Desconectarse del sistema.
RF3: Conectar a un proceso.
RF4: Desconectar usuario de proceso.
RF5: Mostrar listado de procesos activos.
RF6: Mostrar listado de usuarios conectados a un proceso.
RF7: Gestionar rol de administrador.
RF8: Gestionar tipos de procesos.
RF9: Configurar un modelo de proceso.
RF10: Mostrar un listado de los modelos de procesos inactivos por tipo de proceso.
RF11: Gestionar registro de usuarios.
RF12: Mostrar listado de usuarios conectados.

Requisitos que se cumplieron parcialmente
RF13: Actualizar configuración del NVP.
RF14: Simular los procesos industriales en tiempo real.

RF15: Mostrar el resultado de la simulación de forma gráfica.
RF16: Mostrar el resultado final de la simulación.
RF17: Activar proceso automáticamente.
RF18: Programar el tiempo de duración de la activación.
RF19: Gestionar modelos de procesos.

Requisitos que no se cumplieron
RF20: Limitar el número de usuarios conectados al sistema.
RF21: Limitar el número de usuarios conectados por procesos.
RF22: Limitar el número de procesos activos.
RF23: Establecer conexión con el controlador.
RF24: Gestionar controladores.
RF25: Definir prioridades por tipo de proceso.
RF26: Imprimir informes.
RF27: Mostrar listado de los procesos activados por un usuario.

Requisitos que refieren nuevas funcionalidades
RF28: Mostrar listado de procesos activados en una fecha.
RF29: Mostrar listado de usuarios conectados en una fecha.
RF30: Mostrar listado de procesos activados por un usuario en una fecha.
RF31: Configurar controlador.

2.2.2 REQUISITOS NO FUNCIONALES

Los requisitos no funcionales son las cualidades que se imponen al proyecto y las exigencias de usar un cierto lenguaje de programación o plataforma tecnológica. Un requisito no funcional es una característica ya sea del sistema, del proyecto o del servicio de soporte, que no es requerida junto con la especificación del sistema pero no se satisface añadiendo código, sino cumpliendo con esta como si de una restricción se tratara.

A continuación se detallan los requisitos no funcionales del sistema, los cuales están expresados en lenguaje natural. Se incluyen además, los requisitos no funcionales que permiten la ejecución correcta y eficiente de cada funcionalidad. Los mismos serán identificados con las siglas **RNF** más el **Número del requisito** (Ej. RNF1) y los requisitos.

Requerimientos de Apariencia o Interfaz externa:

RNF1: Diseñado para la resolución de 1024x768, aunque deberá verse en 800x600 o cualquier resolución superior a esta.

RNF2: El sistema tiene que ofrecer una interfaz amigable y fácil de operar.

Requerimientos de Usabilidad:

RNF3: Asimilación del sistema por parte del cliente.

RNF4: La información y las funcionalidades estarán disponibles para todos los usuarios.

Requerimientos de Rendimiento:

RNF5: El sistema deberá ser capaz de gestionar toda la información y dar respuestas a las solicitudes lo más rápido posible.

Requerimientos de Soporte:

RNF6: La aplicación brindará la posibilidad de corregir los defectos en un limitado gasto de trabajo, así como la mantenibilidad y la escalabilidad.

Requerimientos de Portabilidad:

RNF7: La aplicación deberá correr sobre cualquier Sistema Operativo.

Requerimientos de Seguridad:

RNF8: Cada usuario accederá únicamente a la información que le corresponda según su nivel de acceso.

Requerimientos de Software:

RNF9: Sistema Operativo Linux o Windows, gestor de base de datos para el servidor.

RNF10: Sistema Operativo Windows o Linux para las computadoras clientes.

Requerimientos de Hardware:

RNF11: Para el servidor: Pentium IV o superior con 1Gb de RAM o más, microprocesador de 3Hz, disco duro con 3Gb espacio libre.

RNF12: Para las PC clientes: Pentium IV o superior con 512 MB de RAM o más, microprocesador de 2GHz.

Restricciones para el diseño e implementación:

RNF13: Se utilizará la herramienta de modelado Visual Paradigm.

RNF14: Se utilizará como lenguaje de programación C++ e IDE QtCreator.

Requerimientos de Red:

RNF15: Las transacciones y recuperación de los datos en la comunicación servidor - PC cliente, se realizará a través del protocolo TCP/ IP.

RNF16: Protocolos de comunicación con controladores: TCP/IP, DCOM, DDE y OPC.

La tabla 2.1 muestra resultados obtenidos en el flujo de trabajo de requerimientos después de realizado el análisis a los mismos.

Tabla 2.1 Resultados obtenidos en el análisis a los requisitos.

Requisitos	Total
Funcionales	31
No funcionales	16
Requisitos en total	47
Se cumplieron parcialmente	7
No se cumplieron	8
Nuevos requisitos	4
Requisitos que debe cumplir	19

2.3 ANÁLISIS CRÍTICO A LA ARQUITECTURA

La arquitectura es el esqueleto o base de una aplicación, donde se analiza la aplicación desde varios puntos de vista. En ella aparecen los artefactos más importantes y diferentes para establecer un esquema de cómo deben ser los próximos artefactos a construir. De obtenerse un artefacto demasiado diferente de los demás formaría parte de la arquitectura. (Jacobson, 2000).

En las versiones anteriores del NVP, se escogió la arquitectura en tres capas. Por las ventajas que presenta el uso de la misma, como es la organización del código, la reutilización y la flexibilidad, se decidió reutilizar.

Capa de presentación: Es la que interactúa directamente con el usuario, captura la información entrada (realiza un filtrado previo para comprobar que no hay errores de formato) y hace las peticiones a la capa inferior mostrando al usuario la respuesta proveniente de esta. Únicamente se comunica con la capa de negocio.

Capa de negocio: Está conformada por el subsistema Gestión, el cual se ajusta a los requisitos y casos de uso arquitectónicamente significativos. Desde el punto de vista de diseño, esta capa es contenedora de las clases entidades y controladoras. Únicamente se comunica con la capa de Acceso a Datos.

Capa de Acceso a Datos: Contiene clases que interactúan con la base de datos y permiten, utilizando los procedimientos almacenados generados, realizar todas las operaciones con la base de datos de forma transparente para la capa de negocio. (Gámez, 2010)

2.4 REESTRUCTURACIÓN DEL MODELO DEL SISTEMA

El modelo del sistema está basado en las funcionalidades que este debe cumplir. Se realiza con el objetivo de ayudar a los clientes, usuarios y desarrolladores a llegar a un acuerdo sobre cómo utilizar el sistema. La mayoría de los sistemas tienen diversos tipos de usuarios, que son representados mediante los actores. Los cuales a su vez, utilizan el sistema al interactuar con los casos de uso, que no son más que fragmentos de funcionalidad del sistema. Todos los actores y casos de uso del sistema, con sus respectivas descripciones forman un Modelo de Casos de Uso del Sistema (Jacobson, 2000).

El Modelo de Casos de Uso del NVP incluye tres actores, el actor Usuario, Usuario Maestro y el actor Administrador. Estos se describen a continuación.

Tabla 2.2 Actores del sistema.

Actor	Descripción
Usuario	Este actor representa a todos los usuarios que trabajan con la aplicación.
Usuario Maestro	Este actor es una especificación del actor “Usuario”, y es el primer usuario que se conecta a un proceso, puede ejecutar todas las funcionalidades igual que el actor usuario, además de configurar y activar un proceso.
Administrador	Este actor es una especificación del actor “Usuario Maestro”, es el encargado de controlar el buen funcionamiento de la aplicación, puede ejecutar todas las funcionalidades igual que el actor usuario maestro, además de que posee usuario y contraseña.

2.4.2 Diagrama de Casos de usos del sistema

Durante el refinamiento realizado a los requisitos planteados inicialmente para el NVP surgieron cambios los cuales se describen en la especificación de cada caso de uso y se exponen en el diagrama de casos de usos del sistema para ofrecer una visión general de las principales funcionalidades que serán implementadas en el mismo, destacándose en verde azul los casos de usos que van ser rediseñados e implementados y los que hacen referencia a las nuevas funcionalidades.

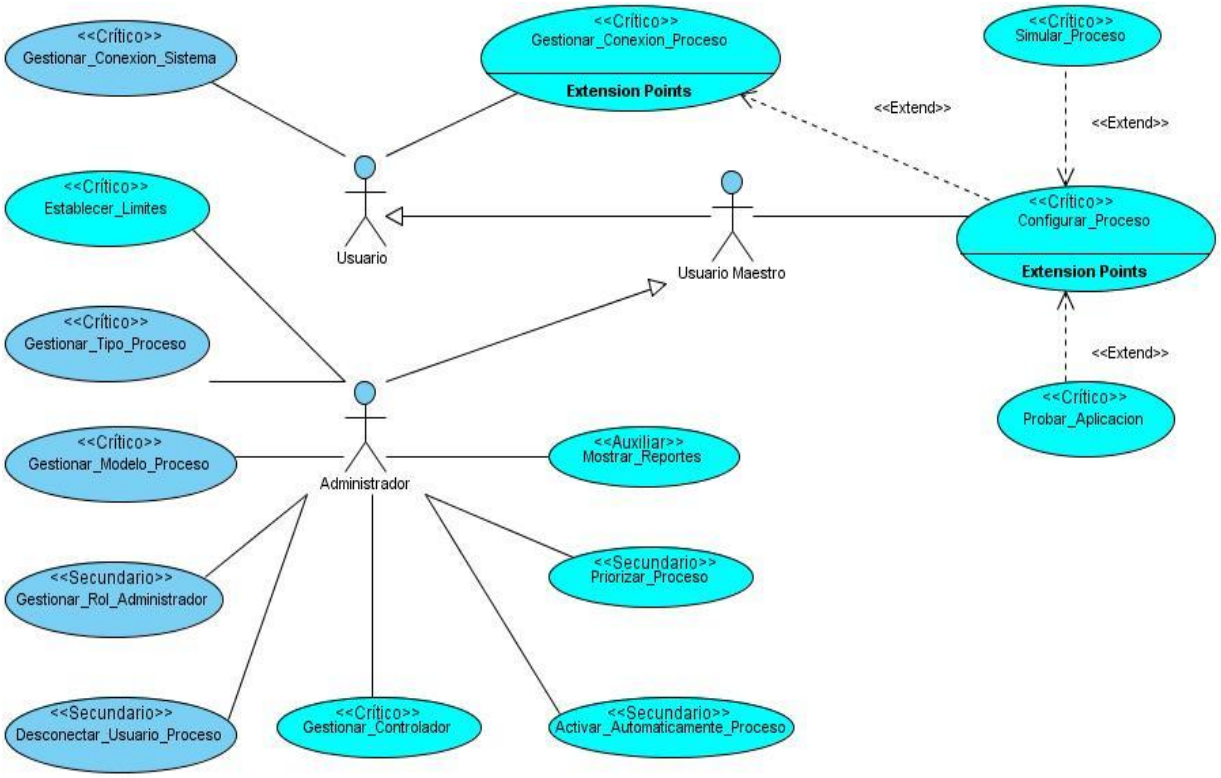


Figura 2.1 Diagrama de Caso de Uso del Sistema

2.4.3 ESPECIFICACIÓN DE LOS CASOS DE USOS DEL SISTEMA

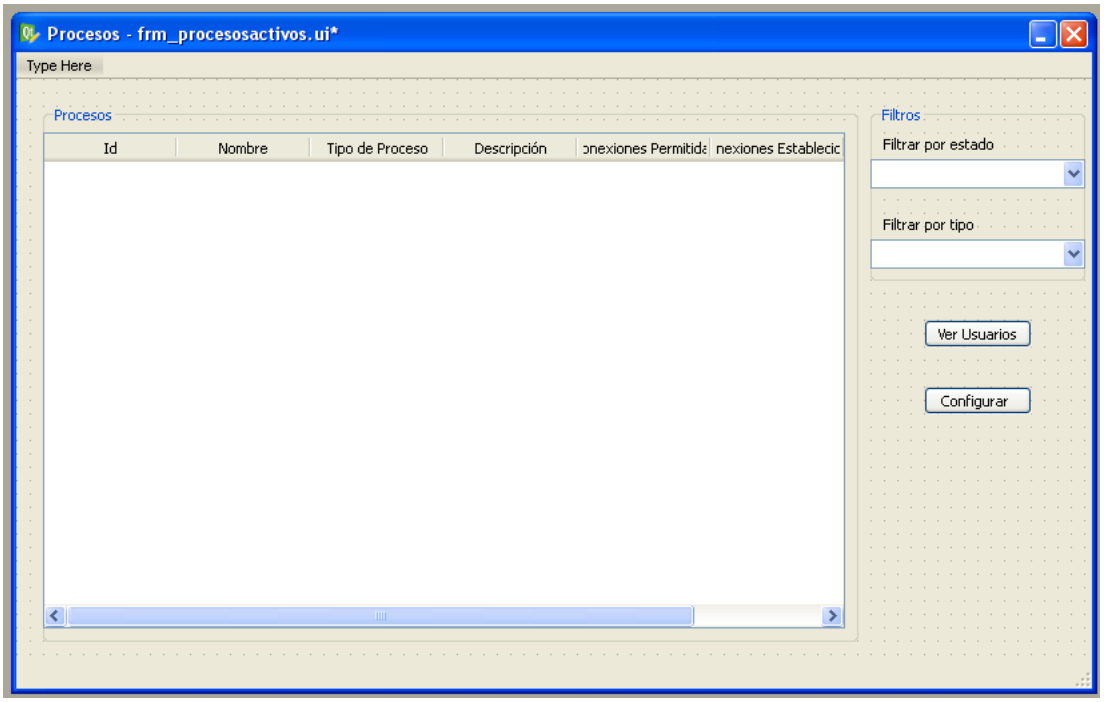
Las especificaciones de casos de uso se realizaron con el objetivo de lograr una mejor comprensión de los procesos a automatizar. En ellas se describen los diferentes flujos de sucesos entre el actor y el sistema, auxiliados de prototipos de interfaz de usuario no funcionales. A continuación se presenta la descripción del caso de uso crítico del sistema *Configurar_Proceso*, las restantes descripciones se pueden consultar en el documento “Modelo de Sistema”.

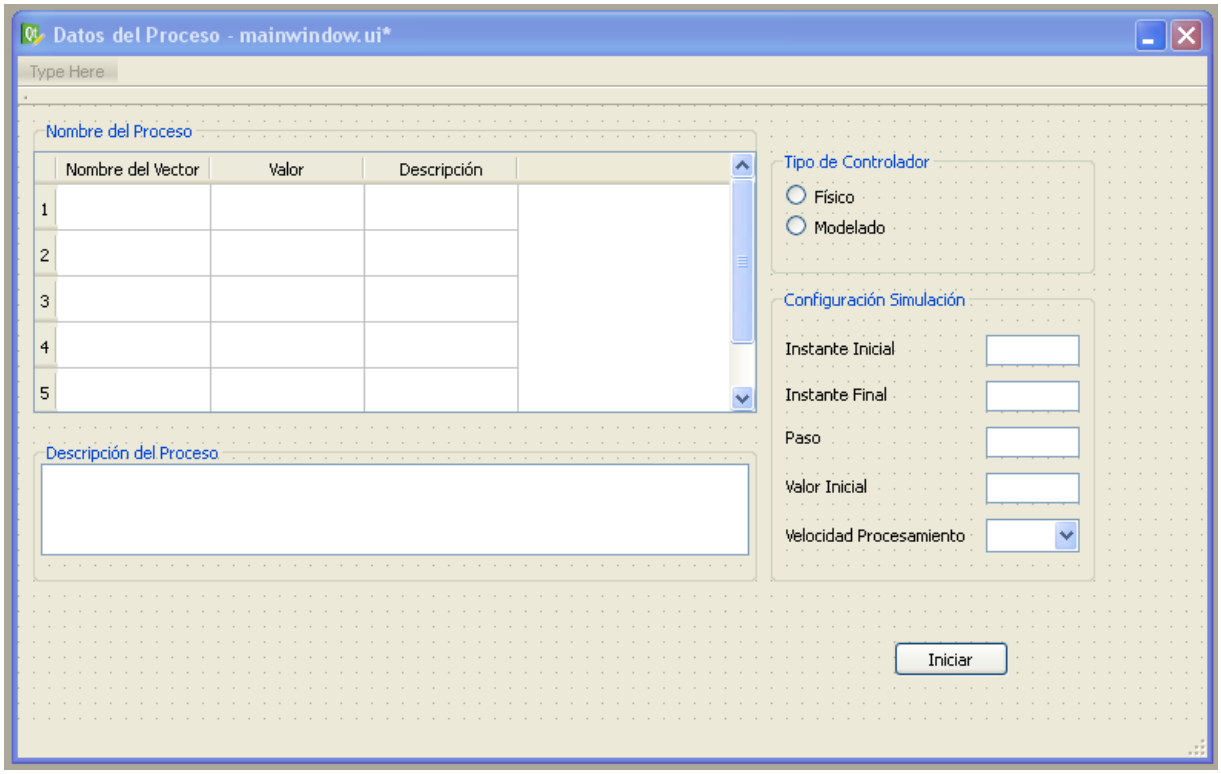
Caso de Uso:	Configurar un Proceso (Extendido de Gestionar_Conexión_Proceso)
Actores:	Usuario Maestro

Resumen:	El caso de uso se inicia en el flujo de acciones del caso de uso Gestionar_Conexión_Proceso cuando el usuario escoge un proceso no iniciado para su simulación, el usuario maestro configura el proceso introduciendo los datos necesarios, el sistema realiza la acción seleccionada y finaliza el caso de uso.
Precondiciones:	El cliente debió autenticarse como administrador o haber seleccionado ser usuario maestro, además el sistema debió verificar los límites para la simulación del proceso seleccionado.
Referencias	RF9
Prioridad	Crítico

Flujo Normal de Eventos	
Sección "Configurar Proceso"	
Acción del Actor	**Respuesta del Sistema**
2.1 El Usuario Maestro o el administrador entra los datos.	1. Cuando se inicia el caso de uso Gestionar_Conexión_Proceso se escogió la opción de conectarse a un proceso inactivo y el sistema verificó la capacidad, se activa el caso de uso Configurar_Proceso y se muestra la interfaz para configurar el proceso. 2. El sistema localiza los datos necesarios para configurar el proceso y muestra la interfaz para entrar los datos. 3. El sistema verifica los datos entrados. Si los datos

	son incorrectos ver flujo alterno ¨Datos incorrectos¨. 4. El sistema verifica si se escogió algún tipo de controlador. Si no se escogió ningún controlador, se activa el caso de uso Simular_Proceso. Si se escogió un controlador ver flujo alterno ¨Seleccionó controlador¨





Flujos Alternos	
"Datos Incorrectos"	
Acción del Actor	**Respuesta del Sistema**
	3. El sistema muestra un mensaje los ¨Datos introducidos son incorrectos¨ y regresa al paso 2.
Flujos Alternos	
"Seleccionó Controlador"	

Acción del Actor	Respuesta del Sistema
	4. Se activa el caso de uso Probar_Aplicación.
Poscondiciones	El proceso queda configurado.

2.5 REESTRUCTURACIÓN DEL MODELO DEL ANÁLISIS

Durante el análisis, se analizaron los requisitos que se describen en la captura de requerimientos, refinándolos y estructurándolos. El objetivo de hacerlo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que ayude a estructurar el sistema, incluyendo su arquitectura. (Jacobson, 2000)

El modelo del análisis es un artefacto que usualmente se genera para entender claramente los requisitos y realizar mejor el diseño. Es usado para representar la estructura global del sistema, describe la realización de casos de uso y sirve como una abstracción del Modelo de Diseño.

2.5.1 DIAGRAMAS DE CLASES DEL ANÁLISIS

En el diagrama de clases de análisis se representan los conceptos fundamentales que comprende el dominio del problema. Estos diagramas muestran que clases participan en las realizaciones de los distintos casos de uso, representan las definiciones y relaciones entre las clases. Las clases del análisis se clasifican en Interfaz, de Control o Entidad.

Clase Interfaz: Modela la interacción entre el sistema y sus actores.

Clase Control: Coordina la realización de uno o unos pocos casos de uso coordinando las actividades de los objetos que implementan la funcionalidad del caso de uso.

Clase Entidad: Modela la información que posee larga vida y que es a menudo persistente.

La siguiente imagen muestra el diagrama de clases del análisis correspondiente al caso de uso *Configurar_Proceso.* Para consultar los diagramas de los restantes casos de uso consultar el documento "Modelo del Análisis".

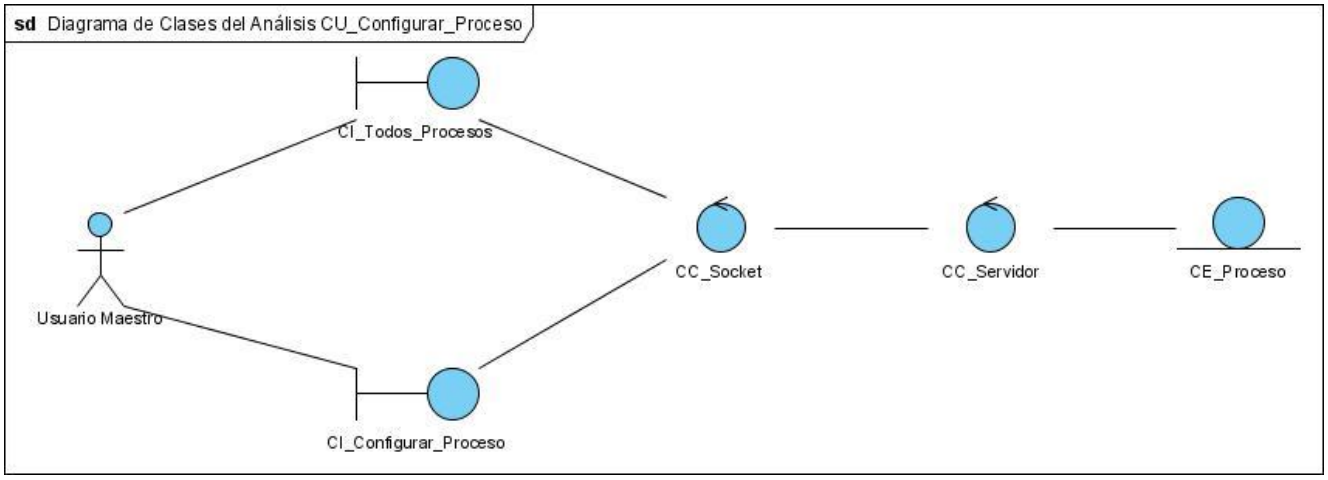


Figura 2.2 Diagrama de Clases del Análisis del CU_Configurar_Proceso

2.5.2 DIAGRAMAS DE INTERACCIÓN

Los diagramas de interacción son diagramas que describen como un grupo de objetos colaboran para conseguir algún fin. Estos diagramas muestran objetos, así como los mensajes que se pasan entre ellos dentro del caso de uso. Los diagramas de interacción capturan el comportamiento de los casos de uso y tienen dos formas de expresarse, los diagramas de secuencia y de colaboración.

Diagramas de secuencia: Muestran las interacciones expresadas en función de secuencias temporales. Destaca el orden temporal de los mensajes.

Diagramas de colaboración: Muestran las relaciones entre los objetos y los mensajes que intercambian. Destaca la organización estructural de los objetos que envían y reciben mensajes.

La siguiente imagen expone el diagrama de colaboración del análisis del caso de uso *Configurar_Proceso*. Para consultar los diagramas de colaboración del análisis correspondiente al resto de los casos de uso del sistema ver el documento "Modelo del Análisis".

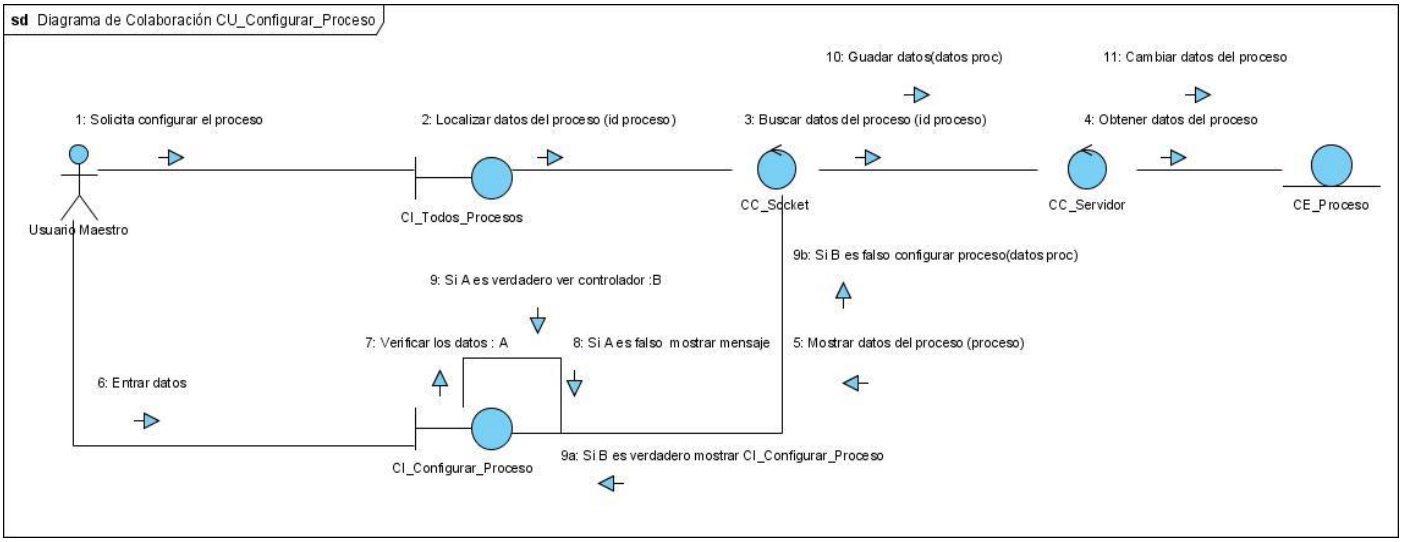


Figura 2.3 Diagrama de colaboración del CU_Configurar_Proceso

2.6 REESTRUCTURACIÓN DEL MODELO DEL DISEÑO

El diseño es un refinamiento del análisis que tiene en cuenta los requisitos no funcionales, o sea como cumple el sistema sus objetivos. El modelo de diseño está muy próximo al de implementación, obviamente para guardar y mantener el modelo de diseño a través del ciclo de vida completo del software. Es un modelo de objetos que describe la realización de los casos de uso. Es una entrada esencial a las actividades de implementación y prueba.

En la versión a desarrollar se reutilizó parte del diseño de clases empleado por la versión anterior, debido a su robustez y su eficiencia desde el punto de vista de los requisitos que debe cumplir el sistema.

El subsistema Simular-Graficar fue repartido entre los dos anteriores, debido a que se decidió dejar la simulación como responsabilidad de las clases gestoras, y la graficación de los datos quedaría como tarea de la presentación. El subsistema Gestión fue reutilizado para la nueva versión. La inserción de los lazos cerrados de los controladores modelos y físicos estarán contenidos en el subsistema Gestión al igual que el módulo de reportes a incorporarle en la nueva versión del NVP, quedando el modelo de diseño de la siguiente forma:

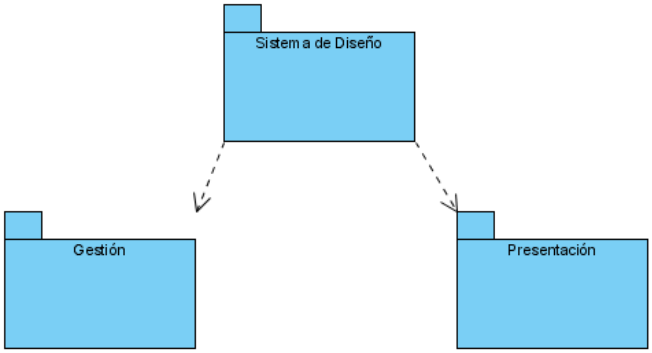


Figura 2.4 Modelo de Diseño del Sistema.

Para garantizar una mayor comprensión del nuevo modelo de diseño del sistema a implementar se describe de forma breve los subsistemas por los cuales va a estar compuesta la aplicación.

Subsistema Gestión: Contiene las clases que intervienen en el proceso de autenticación y registro de los usuarios al sistema. Además, permite la conexión a las diferentes partes de la aplicación y a los procesos. En este subsistema se gestiona toda la información necesaria para el correcto funcionamiento del software. El sistema de reporte a desarrollar en esta versión del NVP va a estar contenido dentro de este subsistema para un mayor entendimiento por parte de los desarrolladores y va estar conformado por dos clases controladoras, CC_Socket y la CC_Servidor, que va a utilizar las entidades de este subsistema.

Subsistema Presentación: Este subsistema contiene las interfaces de usuario que se generan.

2.6.1 DIAGRAMAS DE CLASES DEL DISEÑO

Los diagramas de clases del diseño son los diagramas principales en el flujo de trabajo análisis y diseño. En cada uno se especifica la estructura de clases del sistema y sus relaciones. Durante el análisis del sistema, los diagramas se desarrollan buscando una solución ideal. Durante el diseño, se modifica esos diagramas para satisfacer los detalles de las implementaciones.

En la figura siguiente se refleja el diagrama de clases del diseño para el caso de uso *Configurar_Proceso*. Para conocer los diagramas de clases del diseño correspondientes al resto de los casos de uso del sistema consultar el documento “Modelo de Diseño”.

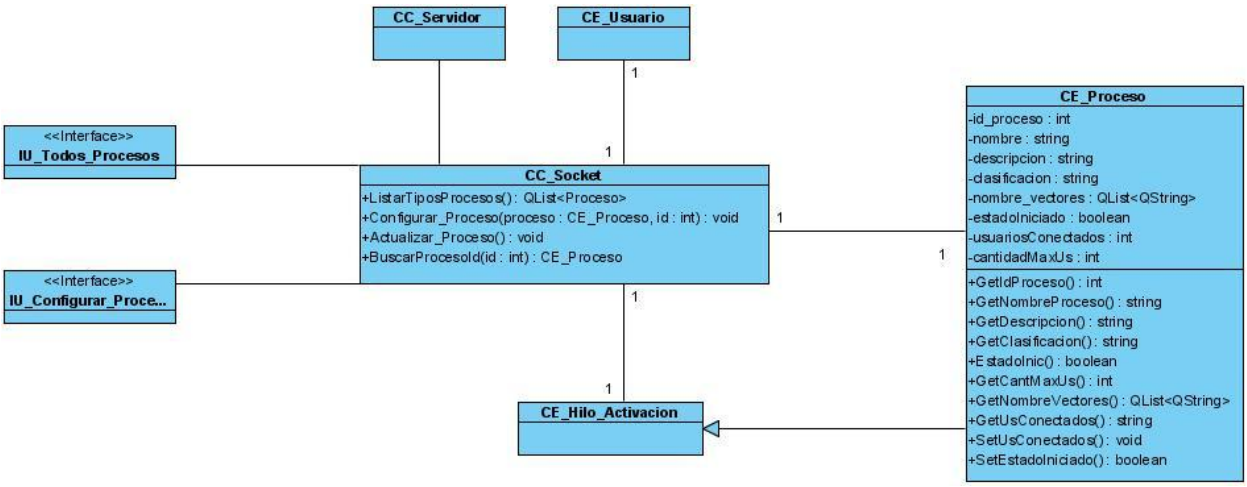


Figura 2.5 Diagrama de clases del diseño CU_Configurar_Proceso

2.6.2 PATRONES DE DISEÑO EMPLEADOS EN LA SOLUCIÓN

Los patrones de diseño proponen soluciones simples y exitosas a problemas habituales que se pueden presentar durante el diseño, basadas en la experiencia. Un patrón de diseño soluciona problemas que existen en muchos niveles de abstracción. Hay patrones que abarcan las distintas etapas del desarrollo desde el análisis hasta el diseño y desde la arquitectura hasta la implementación.

El catálogo de patrones más famoso es el contenido en el libro *Design Patterns: Elements of Reusable Object-Oriented Software*, agrupa los patrones en categorías de acuerdo con su propósito.

En el NVP se usaron algunos de estos patrones en el diseño de la solución los cuales se describen a continuación:

Patrones Creacionales: Fábrica Abstracta (Abstract Factory)

Una interfaz en programación es una clase que define el comportamiento de un objeto sin describir específicamente como lo va a hacer, para este caso se crean objetos que son del tipo Singleton. La característica más apremiante es que tiene un amplio espectro de objetos que deben crearse con estas características. Por tanto, se usa esta interfaz con el objetivo de crear una fábrica de objetos que sean

persistentes en memoria (Objetos singletons), para esto se define un método abstracto que se llama newObject (QString className); y se le pasa por parámetro un string que sería el tipo de datos que se va a usar. La clase ObjectFactory al extender de ella debe reimplementar el método abstracto (virtual) newObject (QString) y es en la clase ObjectFactory donde se crean los métodos.

Patrón Prototipo (Prototype)

Es un patrón de creación, cuyo objetivo es especificar los tipos de objetos a crear por medio de una instancia que hace de prototipo, creando nuevos objetos copiando dicha instancia; es decir, se basa en clonar un prototipo dado. Específicamente en el NVP, cuando un usuario inicia un nuevo proceso se hace una copia del modelo existente utilizando este patrón, manteniéndose la configuración que el administrador del sistema le estableció con anterioridad.

Patrón Solitario (Singleton)

Con este patrón se garantiza una única instancia de aquellas clases de las que se desee tener una sola en toda la aplicación, proporcionando un punto de acceso global a las mismas. Tiene como ventajas que reduce el espacio de nombres y es una mejora sobre las variables globales. Es usado debido a la necesidad de trabajar con el mismo objeto en distintos momentos y distintos subsistemas. (Escobar, 2008) Específicamente en el NVP, este patrón se utiliza en las clases gestoras y en las fachadas de los subsistemas, garantizando una única instancia de ellas. Esto suele hacerse cuando se aplica el Patrón Fachada, que se explicará a continuación.

Patrón Estructural: Fachada (Facade)

Proporciona una interfaz sencilla y unificada para un conjunto de clases o subsistemas, siendo más fácil de usar. Permite reducir la complejidad y minimizar las dependencias, el acceso de los usuarios a los subsistemas es por medio de la clase fachada. Esta clase es la encargada de reenviar las peticiones a los objetos de los subsistemas, para no acceder directamente a los mismos, ocultando la complejidad de ellos. Este patrón favorece un bajo acoplamiento entre los usuarios y los subsistemas, respondiendo a uno de los patrones GRASP. Va a permitir variar las clases internas, de manera transparente a los usuarios

que las utilizan, favoreciendo de esta forma la división en capas de la aplicación. Estas clases van a utilizar el patrón Solitario lo que va a permitir un acceso global a ellas. (Escobar, 2008)

2.7 REESTRUCTURACIÓN DEL MODELO DE IMPLEMENTACIÓN

El Modelo de Implementación tiene gran importancia, mediante su uso los desarrolladores entienden claramente el funcionamiento del sistema antes de comenzar a escribir las líneas de código. Este modelo está conformado por los Diagrama de Componentes y de Despliegue. Además de que su objetivo fundamental es desarrollar la arquitectura y el sistema como un todo, es decir, describir cómo las clases del Modelo de Diseño se implementan en términos de componentes tales como ejecutables, ficheros de código fuentes o tablas de una base de datos.

2.7.1 DIAGRAMA DE DESPLIEGUE

Los Diagramas de Despliegue muestran las relaciones físicas entre los componentes del hardware y el software, es decir, la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes del software. Un Diagrama de Despliegue es un grafo de nodos, unidos por conexiones de comunicación, donde un nodo puede contener instancias de componentes. Es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye las funcionalidades entre los nodos de cómputo.

En la figura 2.5 se representa el Diagrama de Despliegue del NVP. Como se puede observar la PC Cliente se conecta a través de un protocolo al servidor donde se encuentra la aplicación, de esta forma no se necesita tener instalado el producto localmente. En el caso del servidor central, se tienen instalados un servidor de la aplicación y uno de base de datos, además de los componentes necesarios para el funcionamiento del producto. La impresora es el único dispositivo que se utiliza.

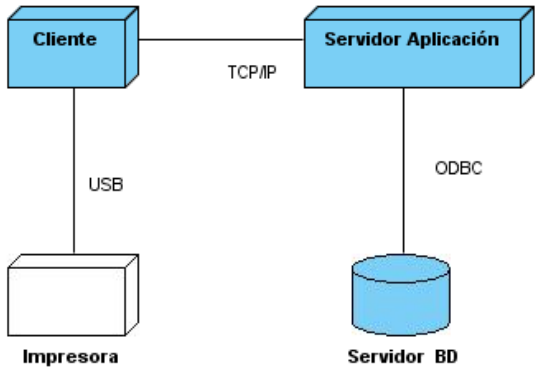


Figura 2.5 Diagrama de Despliegue

2.7.2 DIAGRAMA DE COMPONENTES DEL SISTEMA

Los Diagramas de Componentes muestran tanto los componentes software (código fuente, binario y ejecutable) como las relaciones lógicas entre ellos en un sistema. Los componentes representan todos los tipos de elementos del software implicados en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes o bibliotecas cargadas dinámicamente.

Para el mayor entendimiento del sistema se realizó el diagrama de componentes que agrupa los componentes necesarios para su correcto funcionamiento (Figura 2.6)

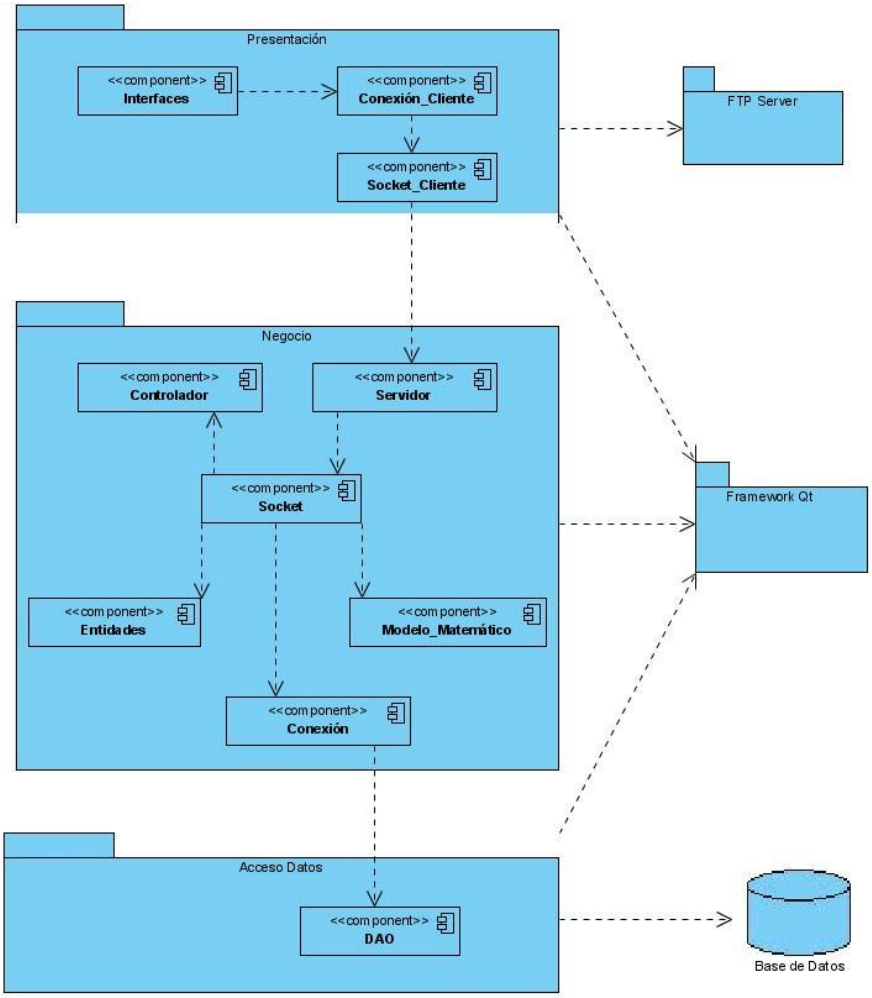


Figura 2.6 Diagrama de Componentes

2.8 HILOS DE EJECUCIÓN

La filosofía del nodo virtual de procesos se fundamenta principalmente en la simulación paralela de varios procesos concurrentes sin que intervengan unos con otros. Una manera de regular el acceso a recursos de hardware exclusivos de un determinado número de consumidores es la virtualización de nodos. En este caso los consumidores son los entornos de ejecución independientes para cada proceso, que se encuentran sujetos a las propiedades de la simulación. (Maier, 2005)

En esta versión se definió cada proceso como un hilo de ejecución, de manera que una vez que se inicia hasta que se detenga, emite valores cada determinado tiempo que son capturados por los sockets de cada usuario. Los sockets son los canales de comunicación que van a ser usados para que los valores de la simulación lleguen a las aplicaciones clientes. Al ser cada hilo independiente y ejecutarse en una dirección de memoria diferente, se garantiza que a la hora de sincronizarlos no exista ningún problema ya que los usuarios maestros y los administradores poseen permisos para detenerlo.

La comunicación de los hilos (procesos) con los sockets se logra mediante la técnica implícita de Qt que conecta signals y slots. Cada vez que un proceso genera un nuevo valor, este se emite mediante una señal y lo captura el slot correspondiente en el socket que enseguida lo manda al socket cliente. Este proceso se hace de manera simultánea para cualquier cantidad de sockets conectados al mismo proceso, lográndose de esta manera la sincronización en tiempo real a los procesos y la simulación concurrente en todos los clientes.

Con la misma arquitectura definida si un proceso, cuando un socket lo pone a correr, no está inicializado, se inicializa y luego se conecta el slot de la clase con los nuevos valores del signal. Si el proceso ya está inicializado solo tiene que conectarse. Por lo que se puede concluir, siempre que se emita un nuevo valor todos los sockets conectados a un mismo proceso lo adquieren al mismo tiempo, garantizando la simulación en tiempo real.

2.8.1 Clases y métodos usados para el trabajo con hilos

El servidor como los clientes del simulador están implementados en C++, utilizando el framework Qt. Debido a esto se hizo uso de las facilidades que estas bibliotecas brindan para la comunicación mediante sockets, como el uso de la clase QThread brindada por QT para el trabajo con hilos.

Para hacer uso de esta funcionalidad se creó una clase llamada Hilo la cual hereda de QThread, permitiendo obtener funcionalidades como start () para iniciar un hilo, terminate () para terminar la ejecución de un hilo, isRunning () para saber si se encuentra ejecutándose. (Gámez, 2010)

2.9 USO DE LAS BIBLIOTECAS DINÁMICAS

Para la adición de nuevos modelos no es necesario modificar el código del NVP. Solamente se debe compilar dichos modelos en bibliotecas dinámicas e indicar al simulador dónde se encuentran, siendo cargadas por este en tiempo de ejecución. En el fichero fuente de cada modelo a adicionar sólo hay que incluir las ecuaciones que describen el comportamiento del proceso a simular y un pequeño número de datos adicionales que ofrecen información sobre dichas ecuaciones. Las ecuaciones representan matemáticamente el modelo. Después de introducida la ecuación es compilada y se le indica al usuario que debe ser guardada en una dirección para luego añadirla al servidor. La simulación consiste en sustituir en las ecuaciones las variables por los valores introducidos por el usuario, luego es resuelta por un método numérico, y la solución encontrada es representada en un gráfico para que el usuario pueda visualizarla.

El modelo es cargado al servidor desde una interfaz gráfica, luego se sube usando el protocolo ftp. Si el modelo se guarda bien en el servidor, entonces se agrega la dirección del modelo a la base de datos. Para poder subir los procesos se configura el server ftp y se le agrega un usuario administrador con una contraseña, que debe ser la misma contenida dentro de la base de datos, con el objetivo de utilizar el objeto que se crea en el cliente además de poseer todos los permisos. Para usar el protocolo ftp se puede emplear cualquier aplicación que simule un servidor ftp. En la nueva versión del NVP se usará el Easy ‘n Quick Ftp Server.

Estos modelos son compilados utilizando el GCC (GNU Compiler Collection o colección de compiladores GNU en español) del sistema operativo GNU/Linux, y su versión para Windows conocida como MinGW (Minimalist GNU for Windows o GNU mínimo para Windows). Para compilar los modelos en bibliotecas dinámicas se emplea el comando gcc -shared -o lib libreria.dll libreria.c. Donde libreria.dll es el nombre especificado de la biblioteca dinámica a obtener, con la extensión .dll para Windows y .so para GNU/Linux; y libreria.c es el archivo de código fuente del modelo de proceso, implementado en el lenguaje C.

2.10 CLASES Y MÉTODOS USADOS PARA LA COMUNICACIÓN CON LA BASE DE DATOS

Las bibliotecas Qt presentan un amplio framework para el acceso a la base de datos. Dicha comunicación puede realizarse directamente utilizando un driver nativo del sistema gestor de base de datos (SGBD) utilizado, o de forma indirecta mediante algún protocolo estándar que garantice dicha comunicación. Estos drivers son integrados a las aplicaciones en forma de plugins. Algunos se deben adicionar, mientras que otros se integran por defecto en la instalación de las bibliotecas Qt. (Muñoz, 2010) Este es el caso de ODBC (estándar seleccionado que fue analizado en el Capítulo 1).

Una de las clases principales que contiene el módulo de acceso a datos es QSqlDatabase. Esta contiene métodos para indicar los parámetros necesarios para establecer la conexión con el SGBD, abrirla y cerrarla. Entre estos se encuentran setHostName, setPort, setDatabase Name, setUserName, setPassword, mediante los cuales se puede especificar la dirección del servidor, el puerto para la conexión, el nombre de la base de datos a consultar, el usuario y la contraseña, respectivamente. Los métodos open y close permiten abrir y cerrar la conexión.

Para crear un nuevo objeto de tipo QSqlDatabase se llama al método estático addDatabase, el cual adiciona la nueva conexión a la lista de conexiones existentes y además la almacena en el objeto creado. Este método tiene varios parámetros, pero el principal es el tipo de conexión a establecer. Es indicado mediante una cadena de caracteres con un código específico para cada tipo de conexión. Es necesario destacar que cada uno de ellos tiene un plugin correspondiente, por lo que para poder ser usado, dicho plugin deberá estar instalado correctamente en la computadora.

En el caso del servidor NVP actual se utilizará QODBC.

El estándar ODBC está integrado por defecto al sistema operativo Windows, facilitando su uso en esta plataforma. En el caso de GNU/Linux se usará unixODBC, herramienta muy similar tanto en su aspecto como en sus funcionalidades a la que presenta Windows. Los drivers necesarios para la conexión entre el estándar ODBC y el gestor de base de datos utilizado, en este caso Access, se pueden encontrar en el sitio oficial del mismo, disponibles tanto para Windows como para GNU/Linux.

2.11 CONCLUSIONES PARCIALES

En este capítulo se realizó la propuesta de solución para el NVP. Para ello se elaboró un conjunto de modelos que garantiza un mayor entendimiento del mismo y describe las principales funcionalidades que podría tener el sistema.

- Se redefinieron los requisitos de la aplicación dándole cumplimientos a las características que debe cumplir el NVP y se reestructuró el diagrama de casos de uso del sistema.
- Se definieron las principales clases del análisis y el diseño mostrando sus relaciones así como la colaboración entre ellas mediante diferentes diagramas.
- Se determinó el modelo diseño del sistema conformado por tres subsistemas: Presentación, Gestión y Simular –Graficar, realizando la descripción de cada uno de ellos y se optó por usar la arquitectura de tres capas escogida en versiones anteriores.
- Para lograr un diseño más legible se utilizaron algunos patrones que garantizan la claridad de los diagramas.
- Para lograr la simulación concurrente se definió un sistema de hilos y se implementó una clase para su manejo y uso.

CAPÍTULO 3: PRUEBAS

3.1 INTRODUCCIÓN

Durante todo el proceso de desarrollo del software se fue realizando la gestión de la calidad del producto mediante el uso de métricas y técnicas de pruebas de software. En este capítulo se mostrarán los resultados obtenidos luego de aplicar las métricas a la especificación de requisitos y al modelo del diseño. Como parte de las técnicas se abordarán las pruebas de caja blanca realizadas a la capa de presentación y a la capa de lógica del negocio, así como las pruebas de caja negra sobre la interfaz del software. Por último se analizarán de los resultados obtenidos durante el proceso de prueba, para conocer si se cumplió con el objetivo propuesto.

3.2 MÉTRICAS PARA LA ESPECIFICACIÓN DE REQUISITOS

Un elemento clave de cualquier proceso de ingeniería es la medición. Las medidas se emplean para entender mejor los atributos de los modelos que se crean. Pero, fundamentalmente, se emplean para valorar la calidad de los productos de ingeniería o de los sistemas que se construyen. (Pressman, 2005)

Para medir la calidad de la Especificación de requisitos de software, se propone una lista de características, algunas de estas son: especificidad (ausencia de ambigüedad), compleción, corrección, comprensión, y la capacidad de verificación. (Overmyer, 1993)

Para determinar la ***especificidad*** (ausencia de ambigüedad) de los requisitos se sugiere una métrica basada en la consistencia de la interpretación de los revisores para cada requisito: (Overmyer, 1993)

$$Q1 = \frac{Rii}{Rt} = \frac{47}{47} = 1$$

Rii: Número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas.

Rt: Total de los requisitos.

La ***compleción*** de los requisitos funcionales puede determinarse calculando la relación:

$$Q2 = \frac{na}{na + nb} = \frac{47}{47 + 0} = 1$$

na: Número de requisitos funcionales completos.

nb: Número de requisitos funcionales pobremente especificados.

Una especificación se considera correcta cuando cada requisito contenido en ella represente una característica que el sistema debe poseer. La ***corrección*** de los requisitos se define usando la siguiente ecuación:

$$Q3 = \frac{Rc}{Rc + Rnv} = \frac{47}{47 + 0} = 1$$

Rc: Número de requisitos que se han validado como correctos.

Rnv: Número de requisitos que no se han validado como correctos todavía.

La ***comprensión*** de los requisitos se determina a partir de la relación que se muestra a continuación.

$$Q4 = \frac{Rbc}{Rt} = \frac{47}{47} = 1$$

Rbc: Número de requisitos que todos los revisores entienden.

La ***capacidad de verificación*** de los requisitos se determina a partir de la relación que se muestra a continuación.

$$Q5 = \frac{nr}{nr + \sum_i c(ri) + \sum_i t(ri)} = \frac{47}{47 + 4{,}4 + 6{,}1} = 0{,}82$$

$\sum_i c(ri)$: Costo para verificar la presencia del requisito i**.**

$\sum_i t(ri)$: Tiempo para verificar la presencia del requisito i**.**

Primeramente para realizar el proceso de medición de la Especificación de requisitos practicándole estas métricas, fue necesario calcular el número total de requisitos. Al realizar la suma de los requisitos funcionales y no funcionales se obtuvieron 47 requisitos en total. Luego de aplicadas estas métricas se concluye, que la Especificación cuenta con la calidad requerida, ya que a medida que los resultados de las métricas se acercan a 1 se alcanza mayor calidad, lo cual contribuye en gran medida a lograr un mejor entendimiento entre clientes y desarrolladores.

Todos los requisitos fueron interpretados de la misma forma, ya que los revisores involucrados en el proceso coincidieron en todas las interpretaciones, cuyo resultado garantiza la ausencia de ambigüedad. Además el alto grado de compleción, a pesar de ser un factor difícil de medir, su valor indica que todos los requisitos que el software debe cumplir han sido incluidos y especificados.

Por otra parte la corrección y la comprensión, obtuvieron un valor óptimo, demostrando que se está en presencia de una especificación que fue bien comprendida por los revisores y que todos los requisitos representan una capacidad o cualidad que debe estar presente en el sistema a construir. Por último, la capacidad de verificación, para la cual también se obtuvo un valor bastante alto, demostrando que cada requisito dentro de la especificación puede ser verificado.

Para un mejor entendimiento de los resultados obtenidos a partir de la aplicación de las métricas, se muestra la gráfica de la Figura 3.1

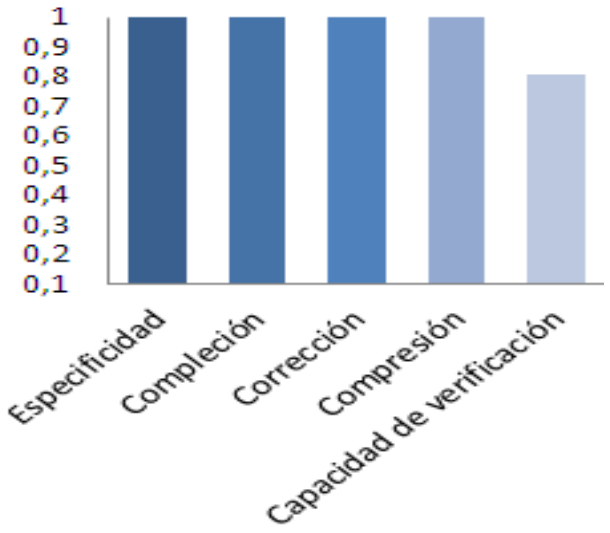


Figura 3.1 Resultados de las métricas aplicadas a la Especificación de requisitos.

3.3 Métricas del Modelo de Diseño

En el mundo del desarrollo del software se han propuesto múltiples conjuntos de métricas, pretendiendo estar centradas en la medición de los diseños y no tanto del código u otros productos del ciclo de vida.

Estas métricas están enfocadas no a la productividad sino a la calidad, por tanto, tratan de evaluar, de manera cuantitativa, si ciertas propiedades supuestamente deseables del diseño se cumplen. Por otro lado, desde la aparición de conceptos como los patrones de diseño (Gamma, 1995) y las refactorizaciones (Fowler, 2000) nuevos elementos para juzgar lo que es un buen o un mal diseño han entrado a formar parte de la literatura especializada en orientación a objetos.

A continuación se mostrarán algunas de las métricas utilizadas en la valoración de los resultados obtenidos en cuanto al diseño del NVP. Estas pueden proporcionarle al diseñador una mejor visión interna y así el diseño evolucionará a un mejor nivel de calidad.

Las **métricas de diseño a nivel de componentes** se concentran en las características internas de los componentes del software. Entre sus medidas se incluye la de cohesión, la cual ayuda al desarrollador de software a juzgar la calidad de un diseño a nivel de componentes. Las métricas presentadas son de "caja blanca" en el sentido de que requieren conocimiento del trabajo interno del módulo en cuestión.

Las métricas de cohesión proporcionan una indicación de la cohesión de las clases del diseño. Mientras más cercano esté el valor de ***CFD*** a 1 mayor será la cohesión del módulo.En el NVP se aplicará la métrica definida para cohesiones funcionales débiles (CFD), la misma tiene como ecuación la siguiente:

$$CFD = \frac{número\ de\ adhesivos(i)}{número\ de\ elementos(i)}$$

Donde *i* se define como la muestra. Número de adhesivos es un elemento que aparece en dos o más clases y súper adhesivo es un elemento que aparece en todas las clases.

$$CFD = \frac{4}{5} = 0{,}8$$

Tabla 3.1 Usabilidad de las clases.

No	Nombre de la Clase	Usabilidad
1	CC_Socket	5
2	CC_Servidor	4

3	CC_Conexion	4
4	CE_Usuario	1
5	CE_Hilo_Activacion	3

La relación del número de clases adhesivas con el número total de elementos de la muestra determina que el diseño de clases posee una cohesión funcional alta para un 80% de fortaleza.

Las **métricas orientadas a clases** hacen hincapié en el encapsulamiento, la herencia, complejidad de clases y polimorfismo. A continuación se describen las que se le aplicaron a los diagramas de clases del diseño. (Pressman, 2005)

Árbol de profundidad de herencia (APH): Mide la longitud máxima desde el nodo hasta la raíz del árbol. Si es demasiado profundo las clases inferiores heredan muchos métodos y hay mucha complejidad en el diseño.

En el Nodo Virtual no fue necesario hacer demasiado uso de la herencia. Aplicando esta métrica al diseño propuesto se obtienen resultados que demuestran su poca complejidad, el árbol de profundidad de herencia toma valor 2, por lo que existe bajo acoplamiento y es de fácil reparación.

Tamaño de la clase (TC): Propone que para determinar el tamaño de una clase se debe tener en cuenta el número total de operaciones y el número de atributos.

El TC es el resultado del promedio general de las dos medidas anteriores para el sistema completo. Si existen valores grandes de TC estos mostrarán que una clase puede tener demasiada responsabilidad, lo cual reducirá la reutilización de la clase y complicará la implementación y la comprobación.

Esta métrica se aplicó para las principales clases de las capas definidas:

Tabla 3.2 Tamaño de clases.

No	Nombre de la Clase	Número de Atributos	Número de Operaciones

1	CC_Socket	4	35
2	CC_Servidor	4	24
3	Conexion	7	33
4	CE_Usuario	4	8
5	CE_Proceso	8	16
6	Hilo_Activacion	8	4
7	Frm_Principal	6	19
8	Metodo_Numerico	5	5

De esta forma, el umbral queda con los datos mostrados a continuación:

Tabla 3.3 Resultado de la métrica ¨Tamaño de clases¨.

Umbral	Tamaño de Clase	Cantidad de Clases
< = 20	Pequeño	5
> 20 y < = 30	Mediano	1
> 30	Grande	2

De acuerdo con los umbrales que se presentan en la tabla anterior, el 62,5% de las clases se consideran pequeñas, el 12,5% son medianas y 25% clases grandes, por tanto, el diseño de clases no es complejo.

3.4 PRUEBAS DEL SOFTWARE

Las pruebas constituyen la actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es realizada de algún aspecto del sistema o componente. La prueba de software es un

elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación. En la comúnmente llamada fase de pruebas se pueden encontrar diferentes niveles y tipos de pruebas, definiéndose como: (Jacobson, 2000)

Pruebas de Integración: Se comprueba la compatibilidad y funcionalidad de los interfaces entre las distintas partes que componen un sistema, estas partes pueden ser módulos, aplicaciones individuales, aplicaciones cliente/servidor, etc.

Pruebas de Validación: Son las pruebas realizadas sobre un software completamente integrado para evaluar el cumplimiento con los requisitos especificados.

Pruebas de Sistema: El software ya validado se integra con el resto del sistema donde algunos tipos de pruebas a considerar son:

- **Rendimiento:** Determinan los tiempos de respuesta, el espacio que ocupa el módulo en disco o en memoria, el flujo de datos que genera a través de un canal de comunicaciones.
- **Resistencia:** Determinan hasta donde puede soportar el programa determinadas condiciones extremas.
- **Robustez:** Determinan la capacidad del programa para soportar entradas incorrectas.
- **Seguridad:** Determinan los niveles de permiso de usuarios, las operaciones de acceso al sistema y acceso a datos.
- **Usabilidad:** Determina la calidad de la experiencia de un usuario en la forma en la que interactúa con el sistema, se considera la facilidad de uso y el grado de satisfacción del usuario.
- **Instalación:** Determinan las operaciones de arranque y actualización del software.

Pruebas de Aceptación*:* Las pruebas que hará el cliente, se determina que el sistema cumple con lo deseado y se obtiene la conformidad del cliente.

Durante el desarrollo del NVP se usaron varios tipos de prueba, entre las cuales se puede mencionar las pruebas de caja blanca que fueron realizadas a las tres capas definidas y a la integración de ambas capas. Las pruebas de caja negra que se llevaron a cabo sobre la interfaz del software.

3.4.1 Pruebas de Caja Blanca

Las pruebas de caja blanca se basan en el diseño de casos de prueba que usan la estructura de control del diseño procedimental para derivarlos.

Mediante las pruebas de caja blanca el ingeniero del software puede obtener casos de prueba que:

- Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
- Ejerciten todas las decisiones lógicas en verdaderas o falsas.
- Ejecuten todos los bucles en sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

Algunos de los métodos empleados en las pruebas de caja blanca son los siguientes:

Prueba del camino básico: Le permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba desarrollados garantizarán que durante la prueba se ejecute por lo menos una vez cada sentencia del programa.

Algunos elementos y conceptos utilizados alrededor de este método son:

Grafo de flujo o grafo del programa: representa el flujo de control lógico de un programa y se utiliza para trazar más fácilmente los caminos de este.

Complejidad ciclomática: es una métrica de software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Representa el número de caminos independientes del conjunto básico de un programa. Esta medida ofrece al probador de software un límite superior para el número de pruebas que debe realizar para garantizar que se ejecutan por lo menos una vez cada sentencia.

Camino independiente: cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición.

Prueba de la estructura de control: Dentro de este tipo de pruebas se contempla el método del camino básico mencionado anteriormente pero además existen otras pruebas asociadas que permiten ampliar la cobertura de la prueba y mejorar su calidad. Estas son: (Pressman, 2005)

- **Prueba de condición:** Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa. Algunos conceptos empleados alrededor de esta prueba son los siguientes:

Condición simple: es una variable lógica o una expresión relacional.

Condición compuesta: está formada por dos o más condiciones simples, operadores lógicos y paréntesis.

- **Prueba del flujo de datos**: Se eligen caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.

- **Prueba de bucle:** Es una técnica que se centra exclusivamente en la validez de las construcciones de bucles (bucles simples, anidados, concatenados y no estructurados).

Pruebas unitarias: En este tipo de pruebas se analiza una porción de código que pueda ser analizada de manera aislada, por ejemplo funciones y métodos. A los que después de pasarle unos parámetros de entrada se deben obtener otros parámetros de salida claramente definidos.

Para dichas pruebas se utilizó el módulo QtTest de prueba ligero que trae Qt, aunque también es conocido como framework QtTestLib, proporcionado por Nokia, para realizar pruebas de unidad en aplicaciones basadas en las librerías Qt, además de extensiones para probar interfaces de usuario. Entre sus características fundamentales se encuentran:

Ligero: Consiste en aproximadamente 6000 líneas de código y 60 símbolos exportados.

Auto contenido: Requiere pocos símbolos de la librería QtCore para pruebas que no son de interfaz.

Probado rápido: No necesita corredores de prueba especiales, ni estar registrado para poder realizar las pruebas.

Prueba repetida de datos: Una prueba puede ser ejecutada múltiples veces con diferentes datos de prueba.

Probado básico de interfaz: QTestLib ofrece funcionalidad para la simulación de ratón y teclado.

Fácilmente extensible: Tipos predefinidos pueden ser fácilmente adicionados a la información del test y a la salida del test. (Nokia, 2009).

En las pruebas realizadas no se detectaron errores en las funcionalidades principales y de mayor uso en la aplicación obteniéndose el resultado que se muestra en la figura 3.2.

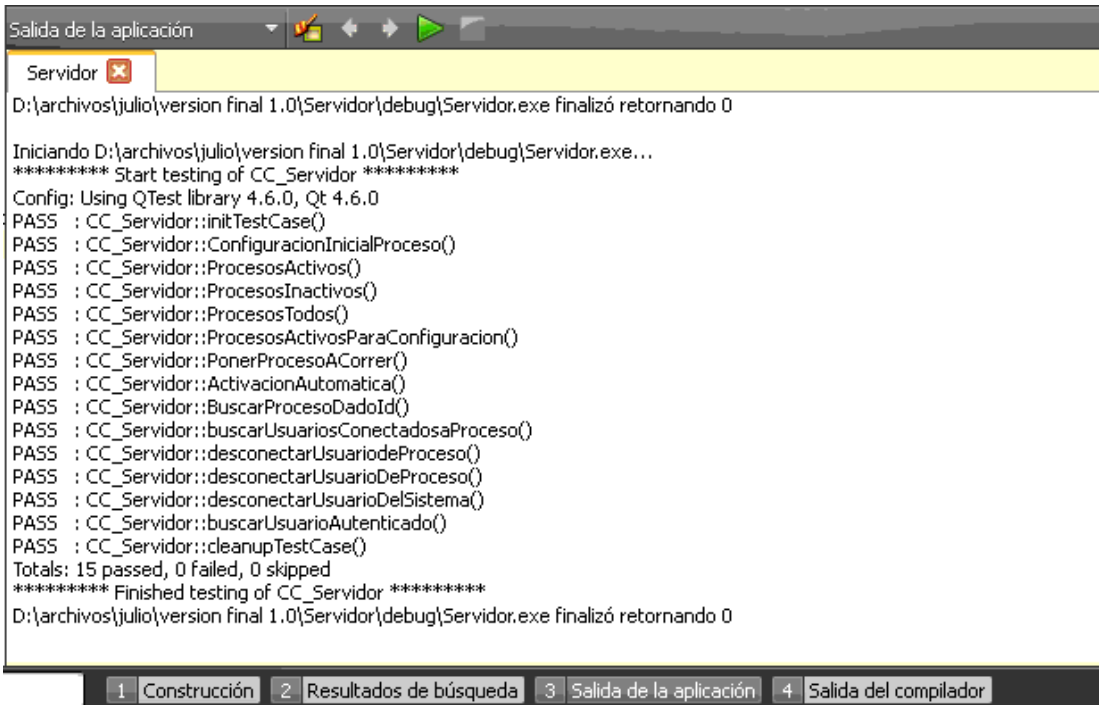


Figura 3.2 Resultado de las pruebas unitarias.

3.4.2 Pruebas de Caja Negra

Las pruebas de Caja Negra parten de los requisitos funcionales, a muy alto nivel, para diseñar pruebas que se aplican sobre el sistema sin necesidad de conocer como está construido por dentro. Estas permiten determinar si la función está siendo desempeñada correctamente por el sistema bajo prueba, aplicando sobre el sistema un conjunto de datos de entrada y observando las salidas que se producen.

Estas pruebas permiten encontrar: (Pressman, 2005)

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las base de datos externas.

- Errores de rendimiento.
- Errores de inicialización y terminación.

Para preparar los casos de prueba hacen falta un número de datos que ayuden a la ejecución de los casos y que permitan que el sistema se ejecute en todas sus variantes. Pueden ser datos válidos o inválidos para el programa según lo que se desea, si es hallar un error o probar una funcionalidad. Los datos se escogen atendiendo a las especificaciones del problema, sin importar los detalles internos del programa, a fin de verificar que el programa se ejecute bien. Para desarrollar la prueba de caja negra existen varias técnicas, entre ellas están: (Pressman, 2005)

Técnica de la Partición de Equivalencia: esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.

Técnica del Análisis de Valores Límites: esta técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.

Técnica de Grafos de Causa-Efecto: es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

Una de las pruebas a desarrollar, utilizando la técnica de caja negra, es la de “Validación”, que es realizada sobre la interfaz del sistema. Esta prueba se realiza luego de integrar todos los módulos que conforman el sistema, para comparar las especificaciones obtenidas durante el análisis y diseño del sistema contra el funcionamiento del sistema final. Para realizar dichas pruebas se diseñaron casos de prueba para cada caso de uso del sistema con el objetivo de encontrar errores en las funcionalidades implementadas.

3.5 Diseño de Casos de Prueba

Con el diseño de los casos de prueba se obtienen un conjunto de pruebas que tienen como misión encontrar defectos y errores al sistema. Cada caso de prueba brinda un número de valores de entradas en

las pruebas, condiciones de ejecución y resultados esperados en las mismas para verificar una determinada funcionalidad del sistema.

El objetivo principal de las pruebas es comprobar que el producto cumpla con las necesidades y exigencias de los usuarios finales del producto. Se diseñaron casos de prueba para cada una de las funcionalidades del sistema los cuales pueden ser consultados en el documento “Casos de Prueba”.

3.6 RESULTADOS

Los resultados obtenidos de las pruebas realizadas al Nodo Virtual de Procesos fueron satisfactorios. Se hicieron pruebas sobre el sistema enfocadas a las técnicas de caja blanca y caja negra. Además, las pruebas del sistema se dividieron en dos etapas. En la primera iteración con ayuda de los casos de prueba se identificaron los errores que tenía el sistema, los cuales fueron reevaluados a través de la realización de nuevas pruebas en una segunda iteración donde no se encontraron errores.

La gráfica que a continuación se presenta muestra los resultados obtenidos en las dos iteraciones de pruebas de validación realizadas al sistema. Incluye los resultados arrojados en las pruebas de caja negra que se realizaron en las capas de presentación, negocio y sobre la interfaz de la aplicación respectivamente.

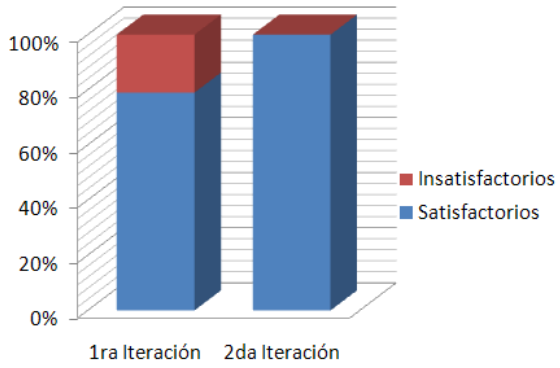


Figura 3.3 Gráfica de los resultados obtenidos en las pruebas al sistema.

Después de la revisión del software por parte de los usuarios finales, se realizó un análisis de los resultados obtenidos teniendo en cuenta los principales aspectos de medición, los cuales se describen a continuación:

Funcionalidad: La aplicación mostró ser funcional, dando solución a los requisitos solicitados por los clientes, los cuales fueron identificados y validados.

Fiabilidad: La aplicación es capaz de detectar cualquier tipo de error, recuperarse y enviar a los interesados el tipo de error ocurrido.

Eficiencia: Debido a la estrategia de integración utilizada entre las capas de presentación y lógica de negocio, la aplicación es capaz de darle respuestas a todas las solicitudes del usuario en un tiempo relativamente corto.

Usabilidad: La aplicación mostró que es muy fácil de usar, aún para personas que poseen pocos conocimientos de los procesos. Además de poseer una interfaz sencilla y agradable.

Mantenibilidad: El sistema cuenta con una arquitectura bastante flexible lo que trae consigo que los cambios sean fáciles de realizar por cualquier persona que posea conocimientos de programación, con vista a mejorar o adicionar nuevas funcionalidades.

3.7 LIBERACIÓN DE LOS ARTEFACTOS OBTENIDOS POR PARTE DE CALIDAD

Para poder tener seguridad en cuanto a la calidad de los artefactos obtenidos, es necesario realizar la medición y revisión de estos. En el proceso de medición de los artefactos “Especificación de los requisitos de software”, “Modelo del sistema”, “Modelo del análisis”, “Modelo del diseño” y el “Modelo de Despliegue”; se llevaron a cabo dos revisiones por parte del proyecto de Calidad del centro CEGEL de la facultad. En la primera se detectaron un promedio de 17 no conformidades, que fueron corregidos satisfactoriamente y así lo constata el acta de liberación. El acta de liberación de la documentación puede ser consultada en el Anexo 1.

3.8 ACEPTACIÓN DEL CLIENTE

Luego de liberados los artefactos anteriormente mencionados y la herramienta implementada *(Ver Epígrafe 3.7)*, fueron presentados a los clientes del sistema, los cuales procedieron a su revisión, con el objetivo de verificar que las especificaciones mostradas cumplieran con todas las expectativas, necesidades, condiciones y propiedades que la aplicación debería contener. Los artefactos y la herramienta presentados fueron aceptados. Como constancia de haber realizado un adecuado proceso de rediseño e implementación del NVP se muestra el acta de aceptación del Nodo Virtual de Procesos por parte del cliente en el Anexo 2.

3.9 CONCLUSIONES PARCIALES

Durante la etapa de pruebas se le aplicaron varias pruebas al software enfocadas a las técnicas de caja blanca y caja negra para comprobar el correcto funcionamiento del sistema y la integración de sus capas. Se utilizó la herramienta **QtTest** integrada al entorno de desarrollo **Qt** para realizar las pruebas de unidad.

Se mostraron los resultados arrojadas en cada una de las iteraciones de pruebas de la aplicación y se comprobó finalmente que el sistema cumplía con los requisitos determinados por los clientes y las normas de calidad establecidas por el Grupo de Calidad de Software en la UCI (**CALISOFT)**.

Luego de aplicar las métricas para evaluar la calidad de la especificación de los requisitos, el diseño arquitectónico y la complejidad de los módulos o subsistemas se concluye que:

- El nivel de ambigüedad en los requisitos disminuyó, lográndose el nivel de especificidad requerida y el entendimiento total por parte de los revisores.
- El 62,5% de las clases que conforman el diseño están dentro de la categoría de pequeñas, lo que demuestra que el proceso de construcción y pruebas del NVP no será complejo.

CONCLUSIONES

Al finalizar el presente trabajo se arribaron a las siguientes conclusiones:

Con el estudio realizado, se identificaron varias herramientas que usan nodos virtuales, pero ninguna de estas herramientas funciona como un nodo virtual de procesos que específicamente simule procesos industriales, constatándose la necesidad de una herramienta interactiva para la simulación de procesos industriales dirigida a los estudiantes de la especialidad de Ingeniería Automática. Esta herramienta fue implementada en cursos anteriores, pero arrojó varias no conformidades lo que dificultó la integración de la misma al proceso de enseñanza de los estudiantes.

El estudio realizado en cuanto a las características que debe cumplir el NVP permitió la selección de las herramientas a utilizar para el diseño y la implementación, la metodología de desarrollo de software, el lenguaje de programación y el estándar de conexión de la base de datos.

El refinamiento de los requisitos permitió identificar las funcionalidades que debe cumplir en la reestructuración del modelo del sistema se obtuvo una descripción más detalla de los casos de usos, así como, de los actores que interactúan con ellos.

En la etapa de análisis y diseño de la aplicación se obtuvo la primera visión de la solución, ya dando respuestas a las no conformidades detectadas e incorporando las funcionalidades no diseñadas.

Se obtuvo un conjunto de modelos que mostró cómo quedaría construida y distribuida la aplicación. El cual fue liberado por el equipo de calidad del centro CEGEL después de haber cumplido todos los aspectos requeridos.

Se implementaron las funcionalidades del sistema definidas en el nuevo diseño propuesto.

Para validar la solución se realizaron varias iteraciones de pruebas de caja blanca y caja negra, auxiliadas por métricas para comprender mejor la calidad del producto y estimar la efectividad de los procesos, mostrando resultados satisfactorios.

Con el desarrollo de este trabajo se le dio cumplimiento al objetivo general de la investigación propuesta: desarrollar una solución informática que elimine las no conformidades detectadas para lograr la incorporación del NVP al proceso docente de los estudiantes de la especialidad de Ingeniería Automática.

RECOMENDACIONES

- Incorporar en la aplicación protocolos de comunicación industrial que permitan la conexión de controladores físicos a la aplicación e incluir la simulación con el uso de controladores físicos.
- Enriquecer la aplicación con nuevos modelos de procesos industriales.

REFERENCIAS BIBLIOGRÁFICAS

Bonaparte, Ubaldo. 2006. Universidad tecnológica nacional. *Facultad regional Tucuman-Departamento de sistemas.* [Online] Ubaldo Bonaparte, Junio 2, 2006. http://www.frt.utn.edu.ar/sistemas/paradigmas/poo.htm.

Cleger, E., Tornés, A. (2009). *"Análisis y Diseño de una herramienta interactiva de simulación de procesos: Nodo Virtual de Procesos. Segunda Iteración."*, tesis en opción al grado de ingeniero en ciencias informáticas, Universidad de las Ciencias Informáticas, La Habana, Cuba.

Cuevas L. (2007). *"Sistema informático de gestión para actividades docentes y extradocentes en la facultad 3. Rol Analista de Sistemas"*, tesis en opción al grado de ingeniero en ciencias informáticas, Universidad de las Ciencias Informáticas, La Habana, Cuba.

Escobar, M., Ortiz, L. (2008). *"Análisis y Diseño de un Nodo Virtual de Procesos."*, tesis en opción al grado de ingeniero en ciencias informáticas, tesis en opción al grado de ingeniero en ciencias informáticas, Universidad de las Ciencias Informáticas, La Habana, Cuba.

Escobedo, Javier García. 2010. JavierGarBedo. *Hilos de ejecución.* [Online] 2010. http://javiergarbedo.es/index.php/apuntes/10-hilos-de-ejecucion/38-hilos-de-ejecucion.

Fowler, M. 2000. *Refactoring: Improving the design of existing code.* s.l. : Addison-Wesley, 2000.

Gámez, Yalice Batista. 2010. *Herramienta interactiva para la enseñanza en la carrera de Ingeniería Automática: Nodo Virtual de Procesos.* La Habana : s.n., 2010.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. 1995. *Design patterns: Elements of reusable object-oriented software.* Boston,MA, USA : Addison-Wesley Longman Publishing Co.Inc, 1995.

Gonce Susana. (2008). *"Arquitectura de un Nodo Virtual de Procesos"*, tesis en opción al grado de ingeniero en ciencias informáticas, Universidad de las Ciencias Informáticas, La Habana, Cuba.

Hernández, Garrigó Daniel and Moreno Martínez., Rolando. 2009. *Solución Informática "Nodo Virtual de Procesos" para la carrera Ingeniería Automática,* tesis en opción al grado de ingeniero en ciencias informáticas, Universidad de las Ciencias Informáticas, La Habana, Cuba.

Hosseinzaman, A and Bargiela, A. 1995. *ADA's Virtual Node base d Water System.* Department of Computing Nottingham Trent University Burton Street:Reino Unido.

Jacobson, I, Booch, G y Rumbaugh, J. (2004) *"El Proceso Unificado de Desarrollo de Software"*. La Habana: Félix Varela.

Jesús, Sanchez Allende. 2005. *Java 2.* s.l. : MCGRAW-HILL, 2005.

López Barrio, C. (2005). *"Metodología de Desarrollo: Programación Extrema".* [En línea]. http://www-lsi.die.upm.es/~carreras/ISSE/programacion_extrema_1.x2.pdf.

Maier S., Herrscher D. (2005*)." On Node Virtualization for Scalable Network Emulation"*, University of Stuttgart, Institute of Parallel and Distributed Systems (IPVS), Germany.

Miyachi T., Chinen K. (2006). *"StarBED and SprinOS: Large scale general purpose network testbed and supporting software"*, Japan Advanced Institute of Science an Technology, Ishikawa, Japan.

Molino N., Bao Z. (2004).*"A Virtual Node Algorithm for Changing Mesh Topology During Simulation",* Stanford University.

Muñoz, Carballo Liosdany and Rodríguez Escalona, Jorge Ernesto. 2010. *Nodo Virtual de Procesos Version 2.0.* Universidad de las Ciencias Informaticas : La Habana : s.n., 2010.

OpenLink, Provider. 2004. *OLE-DB Client Provider.* [Online] 2004. http://docs.openlinksw.com/mt/oledb.html#%28NULL%29.

Overmyer, Davis S. 1993. *Identifying and measuring quality in software requirements specification.* California: Los Alamitos : s.n., 1993.

Palacio, Juan. 2006. NavegaPolis. [Online] 2006. http://www.navegapolis.net/files/s/NST-010_01.pdf.

Pressman, R. (2005). "*Ingeniería del Software. Un enfoque práctico".* La Habana: Félix Varela.

Ripley, Brian. 2010. *ODBC Connectivity.* Department of Statistics, University of Oxford : s.n., 2010.

Rivero A., López E. (2009)."Programación y desarrollo de un Nodo Virtual de Procesos", tesis en opción al grado de Ingeniero en Automática, Instituto Politécnico José Antonio Echeverría, La Habana, Cuba.

Rodríguez F, Lucas and Quintero A, Cabello. 2005. *Simulación de procesos de información y criptografía cuántica.* Departamento Física Aplicada I, Sevilla : s.n., 2005.

Schultz, Rick. 2000. *Using the Oracle ODBC Drivers with Third Party Products.* 2000.

SlideShare. 2009. Visual Paradigm For UML. [Online] 2009. http://www.slideshare.net/vanquishdarkenigma/visual-paradigm-for-uml.

Vaquero J. (2007)."Herramienta interactiva para la enseñanza y entrenamiento en la técnica de control predictivo", tesis en opción al grado de máster, Instituto Politécnico José Antonio Echeverría, La Habana, Cuba.

Welicki, León E. Implementando Extreme Programming en la plataforma. *UPSAM, Departamento de Lenguajes, Sistemas Informáticos e Ingeniería del Software,Madrid España.* [Online] http://www26.brinkster.com/lwelicki/articles/Implementando%20Extreme%20Programming%20en%20la%20plataforma%20NET.pdf.

Wesley, Addison. 2001. *The Annotated C++ Reference Manual.* 2001.

Xavier Albaladejo. 2009. SCRUM un proceso de trabajo. [Online] 8 15, 2009. http://www.proyectosagiles.org/scrum-proceso-trabajo-2-0.

BIBLIOGRAFÍA CONSULTADA

Gámez, Yalice Batista. 2010. *Herramienta interactiva para la enseñanza en la carrera de Ingeniería Automática: Nodo Virtual de Procesos.* La Habana : s.n., 2010.

Pressman, R. (2005). "*Ingeniería del Software. Un enfoque práctico".* La Habana: Félix Varela.

Jacobson, I, Booch, G y Rumbaugh, J. (2004) *"El Proceso Unificado de Desarrollo de Software"*. La Habana: Félix Varela.

Visual Paradigm. 10 Reasons to Choose Visual Paradigm. [En línea] 2011. http://www.visual-paradigm.com/aboutus/10reasons.jsp

Grupo Soluciones Innova.S.A. (GSI). Rational Rose Enterprise. [En línea] 2011. http://www.rational.com.ar/herramientas/roseenterprise.html

Barberán, Manuel. *Funcionamiento del Protocolo TCP-IP.* 1998

Robertson, Suzanne y Robertson, James. *Mastering the Requirements Process.* 2. Portland, Oregon: Addison-Wesley Professional, 2006.

EasySoft. *Linux/Unix ODBC* Oracle Corporation [En línea] 2011 http://www.easysoft.com/developer/interfaces/odbc/linux.html

Katsuhiko, Ogata *Ingeniería de control moderna.* Pearson: 3ra Edición, 1998

GLOSARIO DE TÉRMINOS

Sockets: Es el punto final de un flujo de comunicación bidireccional a través de una red de computadoras basada en protocolo IP.

Signal: Aviso que un objeto puede emitir cuando le ocurre algo (un cambio de estado importante, también denominado evento).

Slot: Método de un objeto que puede ser llamado cuando se genera una señal particular, es decir un signal.

NVP: Nodo Virtual de Procesos.

DLL (dynamic-link library)**:** Biblioteca de vínculos dinámicos. Es un archivo ejecutable que actúa como una biblioteca compartida de funciones.

Hilo: Es un subprograma que se invoca a través de la ejecución de un programa principal y a partir de allí se considera independiente.

Procesos industriales: Es un conjunto de procesos físicos y químicos interrelacionados, implementados por medios físicos. Todo sistema de procesos tiene entradas y salidas. Entradas pueden ser materia prima, temperatura, concentración, presión, etc. Un sistema está sujeto usualmente a señales o perturbaciones que para compensarlas se hace uso de correcciones o acciones de control.

ANEXOS

ANEXO 1: ACTA DE LIBERACIÓN DEL SISTEMA POR PARTE DE CALIDAD

UCI
Universidad de las Ciencias Informáticas

Acta de Liberación de Artefactos, Grupo de Calidad Centro CEGEL de la Facultad 3 de la Universidad de las Ciencias Informáticas.

Miércoles, 8 de junio de 2011.

Luego de haber efectuado _2_ iteraciones de revisiones a los artefactos: Especificación de Requisitos, Modelo de Sistema, Modelo de Análisis, Modelo de Diseño y Modelo de Despliegue de la Herramienta Interactiva para la simulación de procesos industriales Nodo Virtual de Procesos y haberse detectado un promedio de _17_ No Conformidades, se puede afirmar que se han corregido los defectos encontrados, por lo que quedan liberados.

UCI
Universidad de las Ciencias Informáticas
FACULTAD 3

Firma del Asesor y Jefe del Grupo de Calidad Centro CEGEL

Ing. Raúl Velázquez Alvarez

ANEXO 2: ACTA DE ACEPTACIÓN DEL SISTEMA POR PARTE DE LOS USUARIOS FINALES.

UCI Universidad de las Ciencias Informáticas

Acta de aceptación del producto Nodo Virtual de Procesos .

Miércoles 8 de junio del 2011
"Año 53 de la Revolución."

El software de simulación de procesos industriales, Nodo Virtual de Procesos, desarrollado por los diplomantes Julio Jesús García Guevara y Yelina Seija Rodríguez, dio respuesta a las no conformidades detectadas en versiones anteriores. Por estas razones la aplicación reúne todos los requisitos para su aceptación y aplicación en la enseñanza de la especialidad de Ingeniería Automática.

La herramienta interactiva Nodo Virtual de Procesos se considera un medio de enseñanza valioso y necesario en las asignaturas del perfil de Hardware y Automatización. Da respuesta a la carencia de una herramienta que, de manera remota, permita a los estudiantes interactuar con diferentes procesos, configurados por ellos o por su profesor, y con el que puedan probar sus estrategias de control y poner en práctica los conocimientos adquiridos en clase.

UCI Universidad de las Ciencias Informáticas

MsC. Yaricza Gómez Batista
Vicedecana de Formación Facultad 3