Universidad de las Ciencias Informáticas

Facultad 3

Desarrollo de un Cliente OPC para la simulación de procesos industriales en el Nodo Virtual de Procesos

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

Autores:

ADRIÁN SÁNCHEZ PARRA
FRANCISCO GABRIEL RUBIO SANTIESTEBAN

Tutores:

MSC. YALICE GÁMEZ BATISTA
ING. JULIO JESÚS GARCÍA GUEVARA

La Habana, Cuba
Curso 2012-2013

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Francisco Gabriel Rubio Santiesteban

Autor

Adrián Sánchez Parra

Autor

MSc. Yalice Gámez Batista

Tutor

Ing. Julio Jesús García Guevara

Co-Tutor

«Todo en el software cambia, los requisitos cambian, el diseño cambia, el negocio cambia, la tecnología cambia, el equipo cambia, los miembros del equipo cambian; el problema no es el cambio en sí mismo, puesto que sabemos que el cambio va a suceder; el problema es la incapacidad de adaptarnos a dicho cambio cuando éste tiene lugar.»

Kent Beck.

AGRADECIMIENTOS

Adrián Sánchez Parra:

Agradezco a Dios por protegerme durante todo mi camino y darme fuerzas para superar obstáculos y dificultades a lo largo de toda mi vida.

A mi madre por haberme guiado siempre y haber sido madre, padre y amiga. Por darme su amor incondicional, por siempre estar cuando la necesité, porque sin su apoyo no hubiera sido posible este resultado.

A mis abuelos China y Librado por darme su apoyo siempre.

A mis hermanos y amigos que me han acompañado desde la primaria Nilberto, Luis Angel y el Carlos Julio, también a los amigos que eh hecho en el trascurso de estos 5 años en la universidad donde hemos compartido buenos y malos momentos.

A mi compañero de tesis por dejarme ser parte de este trabajo tan grande, por prestarme sus conocimientos, paciencia y voluntad.

A mis tutores por haberme apoyado en la realización de este trabajo.

Gracias a todas las personas que ayudaron directa e indirectamente en la realización de este proyecto, que de una forma u otra hayan hecho posible que hoy esté realizando mi sueño de ser ingeniero.

AGRADECIMIENTOS

Francisco Gabriel Rubio Santiesteban:

Quiero agradecer a todos los que me han guiado en el camino, con su apoyo y voluntad para lograr este título, que más que mío es de ustedes también, porque sin su ayuda no lo hubiese logrado.

Especialmente a mi mamá, mi vieja quien ha logrado ser la persona más preciada en mi vida, que el día que me faltes no sé qué haré sin ti y a mi papá que no estas hoy aquí pero se siente como si estuvieras.

A mis abuelos Jesús y Elena por haberme querido tanto y apoyado en este viaje tan largo y de tantos estudios.

A toda la familia en General, a mis tías lindas, que tanto me complacen, a mis primos por estar siempre ahí jodiendo día, tras día y a los demás de la familia que tanto se preocupan por mí.

A mis amigos de toda la vida, Herbert (La FOCA) el hermano varón que no tengo, a Geiler (La vieja) siempre protestando, Annis, Isa y Yunior, el piquete completo.

Y a los amigos que hicieron la escuela una casa grande, por formar un team inseparable y que logró graduarse pese a todo. Al creador de los PuDiNeS que tanta guerra dieron, el Jordan (El mejor Zerg y el mejor Pudge), al Niva (El terran más Tufao), al July, a Juan, al Eckó, al Fide, a Scorfield y a todos lo que no menciono que están hoy aquí compartiendo con un hermano.

A los amores de mi vida, a todas las quiero mucho y que cuando tengan un hijo sea grande y lindo como yo. A Doris en especial por hacerme una mejor persona y por quererme tanto, a las chicas del aula siempre mis favoritas, un beso.

A mis profesores, en especial a Danaysa, Susana, Aray, Hector, Yohan, Yarina y Yinet.

A mis tutores Julio (Ya eso ta matao) y Yalice (que nos va a desheredar), por lograr que hoy se hiciera posible el sueño de ser Ingeniero.

Gracias a todos los demás que no menciono y porque el documento es de 80 páginas.

DEDICATORIA

Adrián Sánchez Parra:

A mi madre, por ser el pilar más importante y por hacer de mí una mejor persona a través de sus consejos, enseñanzas y amor.

A mi padre, a pesar de nuestra distancia física, siento que estás conmigo siempre y aunque nos faltaron muchas cosas por vivir juntos, sé que este momento hubiera sido tan especial para ti como lo es para mí.

A mis abuelos China y Librado por estar siempre a mi lado apoyándome y aconsejándome.

A mi tío Orlando, a quien quiero como a un padre, por compartir momentos significativos conmigo y por siempre estar dispuesto a escucharme y ayudarme en cualquier momento.

A mis hermanita, porque te quiero infinitamente.

A todo el resto de familia y amigos que de una u otra manera me han llenado de sabiduría para terminar la tesis. A todos en general por darme el tiempo para realizarme profesionalmente.

Francisco Rubio Santiesteban:

Dedicarle este título a todos aquellos que hicieron posible esta realidad hoy, a mi Familia que tanto quería verme graduado.

A mi mamá por estar siempre pendiente de todo, a mi tía Nola y mi tío Gery por ser madre y padre y cuidarme tanto. A mis abuelos que tanto soñaron con verme de ingeniero.

A mis amigos que como yo se gradúan hoy aquí conmigo y a los que faltan. En especial a Luis Manuel, Adrián y al combo letal de 1er año.

A mis tutores para que salga esa maestría y doctorado, que lo que les falta es nada.

Gracias a todos, nos vemos.

Resumen

En el curso 2010-2011 se propuso una herramienta para la simulación de procesos industriales Nodo Virtual de Procesos Industriales (NVP), que permitió crear modelos de procesos industriales, realizar simulaciones, administrar usuarios y procesos, y consultar reportes. Todo con el objetivo de proveer a los estudiantes de la carrera de Ingeniería Automática de una herramienta que les permitiera desarrollar de forma práctica los conocimientos adquiridos en el proceso. Partiendo de esta propuesta inicial, en el presente Trabajo de Diploma se realizó una segunda iteración de la herramienta, que consistió en la incorporación de un estándar de comunicación industrial. Este módulo permite el intercambio de información del sistema con dispositivos de control y supervisión externos, y de esta forma la obtención de resultados más cercanos a la realidad utilizando para ello un cliente OPC.

Para lograrlo se realizó un estudio de los conceptos asociados a simulación y nodos virtuales, el comportamiento de las funcionalidades del estándar de comunicación en otros sistemas similares en el mundo, metodologías de desarrollo de software, lenguajes de programación; además de otros aspectos asociados a la metodología XP(eXtreme Programation). Finalmente se aplicaron métricas y técnicas dirigidas a evaluar la calidad de los artefactos generados, obteniéndose resultados satisfactorios.

ÍNDICE

Introducción

Actualmente las Tecnologías de la Información y las Comunicaciones (TICs) están cada vez más presente en todas las esferas de la sociedad. Las potencialidades que ofrecen en cuanto a la velocidad de procesamiento, difusión y organización hacen que sean cada vez más necesarias e incluso imprescindibles en algunos contextos como la investigación, el desarrollo y la innovación. Esto a su vez presupone el reto de preparar profesionales comprometidos y capacitados para explotar sus bondades.

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, por medio de ella se accede a la capacidad de experimentar independientemente del sistema real, permitiendo a través de la simulación obviar los riesgos inherentes a la experimentación, alcanzando una completa independencia temporal y repitiendo el experimento un número de veces arbitrario. (Gámez, 2010).

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 competentes en esta rama de la ingeniería. Sin embargo, los conocimientos adquiridos por los estudiantes no son suficientes, puesto que no utilizan el uso de herramientas prácticas para enfrentarse a problemas en 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, para resolver esta situación. En la especialidad por su complejidad se recomienda el uso de simuladores, permitiéndole al alumno el trabajo 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.

Como alternativa la Universidad de las Ciencias Informáticas (UCI), en conjunto con desarrolladores del Instituto Superior Politécnico José Antonio Echeverría (CUJAE), han desarrollado una herramienta la cual es capaz de simular procesos industriales. Esta aplicación, también conocida como Nodo Virtual de Procesos Industriales (NVP) permite la simulación simultánea de diferentes procesos concurrentes sin que interfieran unos con

otros. El NVP está dividido en dos partes: un servidor donde se desarrolla la simulación de los procesos y clientes donde trabajan los usuarios y se ven reflejadas las gráficas y datos obtenidos en la herramienta.

Las dos instituciones UCI-CUJAE en conjunto desean agregarle nuevas funcionalidades a la herramienta, de forma tal, que se pueda incorporar al esquema de simulación de dispositivos de control y supervisión, que permitan modelar el comportamiento del sistema a lazo cerrado como si fuera una planta real. Para ello se propone incorporar un estándar de comunicación industrial que posibilite de esta manera la inserción de controladores físicos a las simulaciones permitiéndole a los estudiantes que interactúen con dicha aplicación y obtengan datos confiables que estén más cerca de la realidad, proporcionando mejores resultados por parte de los estudiantes de la CUJAE.

El NVP fue concebido para realizar dos tipos de simulaciones: con controladores físicos y con simulados. Actualmente sólo está implementada la funcionalidad de controladores simulados lo que dificulta la simulación de procesos en un entorno más próximo a la realidad. La **situación problemática** descrita anteriormente, permite concluir que existen insuficiencias en la simulación de procesos industriales a través de la herramienta Nodo Virtual de Procesos ya que no permite la comunicación con controladores físicos que contribuyan a recrear el proceso real.

La incorporación de un protocolo de comunicación industrial al NVP pudiera conducir a simulación del proceso a lazo cerrado más cercana a la realidad justifica la presente propuesta y se formula el siguiente **problema científico**: ¿Cómo facilitar el uso de controladores físicos en la herramienta Nodo Virtual de Procesos utilizando un estándar de comunicación?

Idea a defender: La incorporación de un estándar de comunicación al Nodo Virtual de Procesos contribuirá a simular procesos industriales con controladores físicos, contribuyendo a recrear la realidad.

Objeto de estudio: los estándares y protocolos de comunicación.

Objetivo general: Desarrollar un estándar de comunicación industrial que permita realizar simulaciones de procesos industriales a lazo cerrado con controladores físicos, de forma tal que se contribuya a recrear la realidad.

Campo de acción consiste en los mecanismos y protocolos de comunicación industrial en laboratorios virtuales.

Teniendo en cuenta el problema de investigación y para dar cumplimiento al objetivo de este trabajo se plantean los siguientes **objetivos específicos:**

- Caracterizar los protocolos industriales que contribuya en mayor medida a la incorporación de controladores físicos a partir de la sistematización de los referentes teóricos relacionados con los estándares y protocolos de comunicación industrial.
- Realizar el análisis y diseño del módulo de comunicación industrial.
- Implementar el módulo de comunicación industrial.
- Validar la solución propuesta aplicando diferentes pruebas.

Tareas:

- Elaboración del marco teórico de la investigación
- Estudio de sistemas existentes para resolver problemáticas similares
- Confección de los artefactos del análisis y diseño del estándar de comunicación.
- Implementación del diseño propuesto.
- Selección de las pruebas para la validación de la propuesta.
- Aplicación de las pruebas seleccionadas y constatación de los resultados.

Para la realización de las tareas propuestas se hace uso de algunos de los **métodos de investigación.**

Como **métodos teóricos** se utilizaron**:**

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 los estándares de comunicación, 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: En este capítulo se abordan los elementos previos que permiten lograr un entendimiento del procedimiento a informatizar. También se hace un estudio de los estándares de comunicación, incluyendo metodologías de desarrollo, protocolos de comunicación, plataformas de desarrollo y por último, métodos y técnicas de pruebas de software.

Capítulo 2. PLANIFICACIÓN, DISEÑO E IMPLEMENTACIÓN: En este capítulo se realiza una descripción de la propuesta del sistema y sus principales funcionalidades, se realiza las historias de usuarios, el plan de iteraciones, las tareas de ingeniería y las tarjetas CRC, así como también se refleja la arquitectura del sistema.

Capítulo 3. VALIDACIÓN DE LA SOLUCIÓN: En este capítulo se diseñan y ejecutan los casos de pruebas para probar el correcto funcionamiento de los componentes implementados, se hace un análisis de los resultados obtenidos en las pruebas y se le da solución a las no conformidades detectadas. Se espera que al final de esta investigación se complete la herramienta logrando la utilización de controladores físicos por medio de un estándar de comunicación.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se realiza un análisis profundo de los aspectos técnicos concernientes a la simulación de procesos, la introducción de conceptos básicos y ejemplos para una mayor comprensión. También se realizará un análisis crítico a cerca de los estándares de comunicación. Además se explicarán las herramientas escogidas para dar solución a la problemática propuesta.

1.1 Simulación

El término simular es definido por la Real Academia de Lengua Española como "Representar algo, fingiendo o imitando lo que no es". La etimología de la palabra simulación se deriva del latín *simulatio*, cuyo significado es la acción de fingir o imitar lo que no se es. (REA, 2001).

Esta definición no se ajusta al contexto técnico por lo que para los efectos de esta investigación no permite su comprensión.

Hugh en 1989 expresó: "La simulación refleja un modelo más complejo de la realidad que se pretende experimentar permitiendo una combinatoria de variables más abierta. El grado de libertad es más amplio, ya que, la combinación de variables (cantidad de información) hace que el universo de interacciones y respuestas posibles sea más amplio". (Hugh, 1989)

Tiene como limitante que no refleja el impacto de las TIC para su desarrollo. Lo que si fue abordado por Shannon cuando lo definió: "Simulación es el proceso de diseñar y desarrollar un modelo computarizado de un sistema o proceso y conducir experimentos con este modelo con el propósito de entender el comportamiento del sistema o evaluar varias estrategias con las cuales se puede operar el sistema." (Shannon, 1992)

Shannon por el contrario refleja su concepto a un nivel computarizado de sistemas por el cual se pueden obtener los resultados. Concuerda con Hugh en cuanto al desarrollo de un modelo, no obstante es insuficiente puesto que no tiene en cuenta la duración de la misma, y el tiempo es un recurso crítico.

Teniendo en cuenta estos criterios se asume la definición brindada por Castelán:

"Simulación es una técnica numérica para realizar experimentos en una computadora digital. Involucrando ciertos tipos de modelos matemáticos y lógicos que describen el comportamiento de sistemas de negocios, económicos, sociales, biológicos, físicos o químicos a través de distintos períodos de tiempo." (Castelán, 2006)

1.2 Nodos Virtuales

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.

Basado en estos criterios se desarrolló un Nodo Virtual de Procesos 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, 2003)

Esta filosofía de trabajo permite la simulación simultánea de diferentes procesos concurrentes sin que interfieran uno 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. (Maier, 2003)

De ahí se derivan los siguientes requerimientos para la virtualización del nodo:

- 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.

Teniendo en cuenta estos aspectos se define como nodo virtual de procesos a aquel software que permita implementar modelos de distintos procesos ya sea para su simulación o la prueba de aplicaciones en tiempo real, por lo que será necesario que varios procesos estén activos simultáneamente para requerir de la cantidad de nodos y computadoras. (Ortiz, 2008)

Se establece como nodo al objeto en el cual se interconectan varios elementos. No debe

entenderse como 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.

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 otro u otros nodos. De esta manera, un nodo puede ser el punto de conexión para transmitir los datos, el punto desde el cual se distribuye los datos hacia otros nodos y el punto final al que se transmiten los datos.

Para el establecimiento de pruebas se necesita de una fracción de recursos de un nodo de prueba y un número de aplicaciones dirigidas a dispositivos de pocos recursos de forma tal que se puedan ejecutar varios procesos en este nodo de prueba más conocido como nodo físico o pnodo y que cada proceso provea un entorno de ejecución del mismo de manera separada, a esto se le conoce como nodo virtual o vnodo. (Gámez, 2008)

1.2.1 Herramienta interactiva “Nodo Virtual de Procesos” (NVP)

Teniendo en cuenta las características de los nodos virtuales expuestas anteriormente, la Universidad de las Ciencias Informáticas (UCI) y el Instituto Superior Politécnico José Antonio Echeverría (CUJAE), en conjunto, desarrollaron una herramienta para la simulación de procesos industriales. “

La herramienta consta con un buen refinamiento de los requisitos, una buena estructura del sistema, así como el análisis, diseño e implementación. En esta última fase se garantiza la simulación en tiempo real y funcionamiento de distintos procesos concurrentemente sin que se afecten los unos con los otros. Sin embargo no se completa la herramienta del todo, puesto que la misma debe realizar dos tipos de simulaciones, una con controladores físicos y la otra con modelados, y solo implementa esta última. Por ende es necesaria la utilización de un estándar de comunicación industrial, para establecer la comunicación entre los dispositivos físicos y el sistema.

La implantación de un estándar de comunicación no interfiere en lo absoluto con la arquitectura del sistema, esta arquitectura está compuesta por tres capas, la cual presenta como ventajas la reutilización y flexibilidad del código, haciendo de forma sencilla la integración del estándar.

1.3 Estándares de comunicación industrial

En la industria existen disimiles protocolos y estándares de comunicación. Estos permiten el intercambio de datos entre los diferentes elementos que conforman el lazo de control (planta, sensores, controladores, elementos de acción final, etc.). Entre estos protocolos se destacan el MODBUS, PROFIBUS y OPC.

- **MODBUS**: es un protocolo de comunicación diseñado para permitir a equipos industriales tal como Controladores Lógicos Programables (PLCs), computadores, motores, sensores, y otros tipos de dispositivos físicos de entrada/salida comunicarse sobre una red. La arquitectura que posee es Maestro\Esclavo. Además presenta un alto nivel de código abierto. Las ventajas de que sea tan abierto y conocido también conllevan desventajas como en la integridad, seguridad y confidencialidad de la información, haciendo de MODBUS TCP un sistema de comunicaciones vulnerable con muchas posibilidades de perder información o ver saboteado el sistema por un ataque. (MODBUS, 2001)
- **PROFIBUS**: es un estándar de red de campo abierto e independiente de proveedores, donde la interfaz de ellos permite amplia aplicación en procesos, fabricación y automatización predial. Este estándar es garantizado según los estándares EN 50170 y EN 50254 (PROFIBUS, 2011)

Estos dos estándares tienen como limitante que exigen que todos los dispositivos del lazo usen este mismo estándar para la comunicación. Para el Nodo Virtual de Procesos sería una desventaja ya que el tipo de sistemas con los que se podría experimentar tendrían que estar sujetos a estas restricciones. No obstante para próximas iteraciones no se descarta su incorporación de forma adicional.

- **OPC**: El OPC (OLE for Process Control) es un estándar de comunicación en el campo del control y supervisión de procesos. Este estándar permite que diferentes fuentes de datos envíen datos a un mismo cliente OPC, al que a su vez podrán conectarse diferentes programas compatibles con dicho estándar. De este modo se elimina la necesidad de que todos los programas cuenten con drivers para dialogar con múltiples fuentes de datos, basta que tengan un driver OPC.

De estos estándares se optó por el OPC ya que es un estándar robusto, fiable que integra otros protocolos como DCOM, TCP/IP y DDE. Permite que el sistema pueda comunicarse con una mayor diversidad de dispositivos. El OPC para las necesidades del NVP, se

considera que es el más completo de los estándares de comunicación. Permite además integrar fácilmente la solución de la herramienta NVP, al estándar de comunicación por la estructura que el mismo presenta.

1.3.1 Estándar de comunicación OPC

Los estándares o protocolos de comunicación, todos son privados. Creados por Microsoft en un afán por el intercambio de datos entre aplicaciones de Windows 3.0. Microsoft crea el protocolo DDE (Dynamic Data Exchange), y no pasó mucho tiempo antes de que los usuarios vieran los beneficios de tener su proceso en aplicaciones de uso general como Microsoft Excel. Sin embargo las limitaciones de la DDE se hicieron evidentes, pues no era un protocolo sólido y lo peor de todo su ancho de banda era muy limitado. (OPCCONNECT, 2011)

Cuando en 1992 es lanzado OLE 2.0 (Object Linking Embeding), era evidente que remplazaría todos los usos de DDE. Era más flexible, más robusto y utilizaba mecanismos de transporte más eficientes. (OPCCONNECT, 2011)

En 1994, hubo un interés firme, enfocado a través WinSEM (Windows en Ciencias, Ingeniería y Manufactura), en el uso de técnicas de OLE para mover datos entre las aplicaciones de proceso en (casi) en tiempo real. En particular, un número de vendedores SCADA vio la oportunidad de estandarizar la interfaz entre el núcleo SCADA y los controladores de dispositivo que eran realmente responsable de la adquisición de los datos. (OPCCONNECT, 2011)

En opinión de los que participaban en el esfuerzo WinSEM (Incluyendo Microsoft) se necesitaría un grupo más pequeño y con una fuerte orientación para garantizar la entrega de una norma. En resumen pretendían unirse las diferentes compañías y generar una norma estandarizada del producto. Este fue el origen de la Task Force OPC. (OPCCONNECT, 2011)

La Task Force OPC se hizo pública en el ISA (International Sign Expo) Show 1995 en Nueva Orleans con un comunicado de prensa. Sus miembros consistían de Fisher-Rosemount (ahora Emerson Process Management), Intellution (ahora parte de GE Fanuc), Intuitive Technology (ahora parte de Wizcon Systems), OPTO 22, y Rockwell Software. Microsoft iba a participar en un papel de apoyo y de consulta. (OPCCONNECT, 2011)

La especificación OPC versión 1.0 fue lanzado el 29 de agosto de 1996. Una versión correcta de OPC DA (Data Access Specification), como se conoce ahora, apareció en 1997. (OPCCONNECT, 2011)

Después de haber buscado la opinión en toda la industria, se tomó la decisión de que la especificación OPC debía ser administrado por una organización independiente, sin fines de lucro que se denominará OPC Foundation. (OPCCONNECT, 2011)

La OPC Foundation hizo sentir su presencia en el ISA Show de Chicago 1996, con demostraciones OPC server de varias empresas en el stand de Microsoft, y también la primera reunión de la Asamblea General de socios. (OPCCONNECT, 2011)

Los productos comerciales que utilizan OPC comenzaron a aparecer a finales de 1996. A mediados de 1998, el amplio apoyo a OPC lo había confirmado como el estándar industrial. (OPCCONNECT, 2011)

A partir de este momento se comienzan a desarrollar los principales tipos de especificaciones OPC; 1998 OPC DA 2.0 y OPC AE (Alarms and Events) y en 2000 OPC HDA (Historical Data Access). (OPCCONNECT, 2011)

1.3.1.1 Principales retos de la automatización y simulación, su solución con OPC

Hoy en día la simulación se utiliza de modo generalizado en la comprensión de diferentes tipos de procesos industriales. El mayor reto que enfrentan estas simulaciones con los distintos dispositivos especializados, sistemas de control y aplicaciones es referente a cómo compartir información entre todos estos componentes y el resto de la aplicación.

Antes de introducir qué es OPC y cómo resuelve una de los mayores retos de la automatización y la simulación, se enumerarán los factores que tradicionalmente causaban problemas al compartir información y una breve explicación de cómo OPC soluciona estas situaciones:

- **Obsolescencia de infraestructuras antiguas:** A medida que los fabricantes lanzan nuevos productos, eventualmente dejan de dar soporte a los antiguos. Cuando una nueva versión de (Human Machine Interface) HMI o SCADA ve la luz, es posible que requiera de su propio juego de protocolos que, en ocasiones, dejan de soportar comunicaciones con dispositivos con los que la anterior versión de HMI

o SCADA comunicaban.

OPC extiende la vida útil de sistemas antiguos porque, una vez que se ha configurado un Servidor OPC para el sistema, permite que cualquier aplicación Cliente que utilice OPC pueda comunicarse con el sistema antiguo, sin importar si la aplicación Cliente soporta o no de forma nativa la comunicación con dicho sistema antiguos. Por tanto, OPC permite que aplicaciones Cliente nuevas continúen comunicando con sistemas antiguos.

- **Drivers de comunicación propietarios:** Todas las conexiones punto a punto requerían un protocolo propietario para posibilitar la comunicación entre los extremos específicos. Por ejemplo, si un HMI necesita comunicar con un PLC, se requería de un driver en el HMI escrito específicamente para el protocolo utilizado por el PLC. Si los datos de este PLC necesitaban ser registrados además en un histórico de datos, el programa de registro de datos requería su propio driver porque el driver del HMI sólo se podía utilizar para el HMI y no para el software de registro histórico de datos (que necesitaría otro Driver propietario diferente).

  OPC elimina la necesidad de disponer de drivers específicos entre cada aplicación y la Fuente de Datos. En la Figura #1 a continuación se puede observar, un único protocolo estándar de PLC puede ser compartido simultáneamente por el HMI y la aplicación de registro de datos históricos mediante un conector OPC, con el beneficio añadido de que el conector OPC requiere una única conexión con el PLC, reduciendo así la carga del procesador.

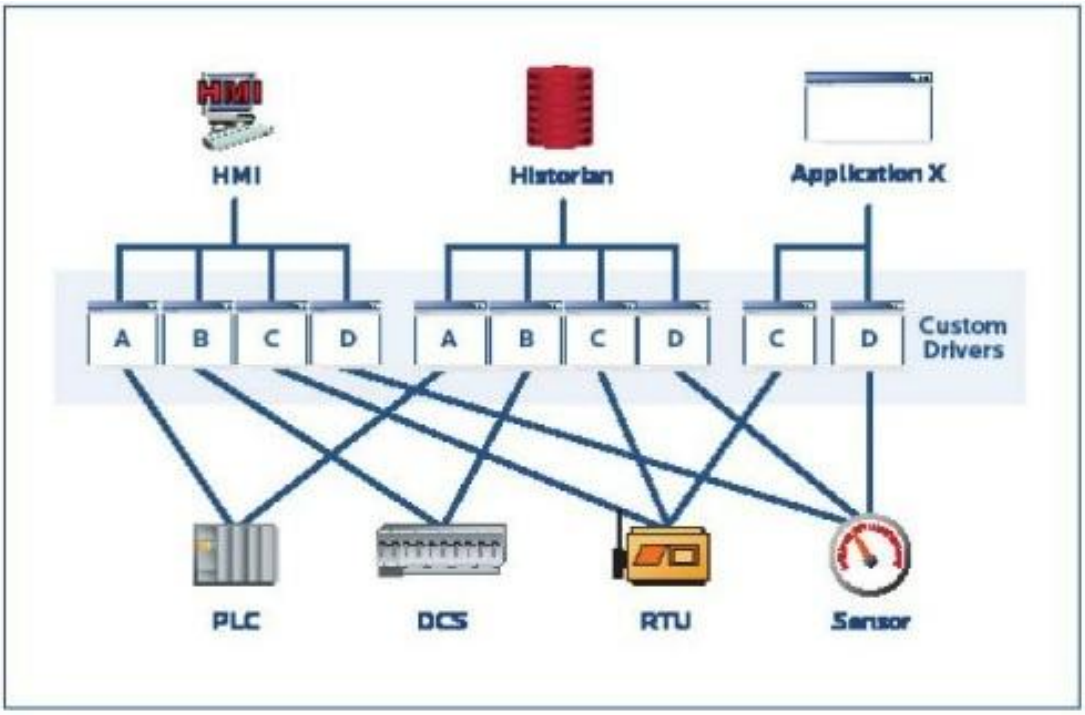

Figura #1: Problema de controladores personalizados – Cada aplicación requiere un dispositivo o un controlador de protocolo específico que le permita comunicarse con los respectivos dispositivos. Los drivers de comunicación no son reutilizables entre aplicaciones, puesto que cada aplicación utiliza su propio formato de datos.

- **Integración compleja:** El uso habitual de protocolos propietarios para cada dispositivo significa que, incluso para un pequeño número de dispositivos y aplicaciones, se requería rápidamente el uso de muchos protocolos. La misma aplicación HMI ejecutándose en múltiples instancias del mismo protocolo en cada ordenador. Si las aplicaciones HMI comunicaban a su vez con dispositivos adicionales, cada HMI requería su propio conjunto de drivers para cada uno de los dispositivos. El mantenimiento de las versiones de las aplicaciones se convertía en una pesadilla.

  Utilizar OPC simplifica enormemente la integración porque, una vez que se configura un Servidor OPC para una Fuente de Datos en particular, todas las aplicaciones que utilizan OPC pueden empezar a compartir datos con esa Fuente de Datos, eliminando la necesidad de Drivers adicionales.

La clave del éxito de OPC en crear comunicaciones auténticamente independientes del fabricante estriba en que OPC abstrae de los detalles de la Implementación de la Fuente de Datos (e. PLC) y Cliente de Datos (e. HMI o SCADA), con lo que los datos se pueden intercambiar entre ellos sin que tengan que saber nada de sus respectivos protocolos de

comunicación nativos y de la organización interna de sus datos como se observa en la Figura #2. Esto, en clara oposición a la aproximación basada en crear aplicaciones basadas en protocolos propietarios que, por definición, son requeridos para comunicar, de forma nativa, la Fuente de Datos con el Cliente de Datos.

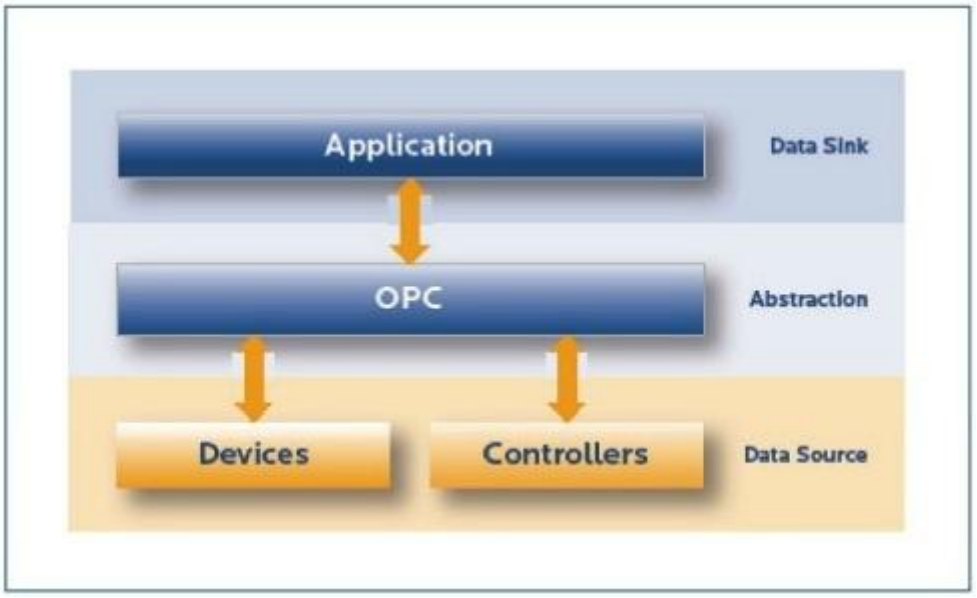

Figura #2: Los Servidores OPC se muestran como un nivel intermedio entre las Fuentes de Datos y el Cliente de Datos, habilitando la intercomunicación sin que ningún lado conozca el protocolo nativo del otro.

1.3.1.2 Funcionamiento del OPC

La abstracción de dispositivo OPC se consigue utilizando dos componentes OPC especializados llamados Cliente OPC y Servidor OPC. Cada uno de ellos es descrito en las siguientes secciones. Es importante resaltar que el hecho de que la Fuente de Datos y el Cliente de Datos puedan comunicar entre sí mediante OPC no significa que sus respectivos protocolos nativos dejen de ser necesarios o hayan sido reemplazados por OPC. Al contrario, estos protocolos e interfaces nativos siguen existiendo, pero sólo comunican con uno de los dos componentes del software OPC. Y son los componentes OPC los que intercambian información entre sí, cerrando así el círculo, para mayor comprensión ver Figura #3. La información puede viajar de la aplicación al dispositivo sin que estos tengan que hablar directamente entre sí.

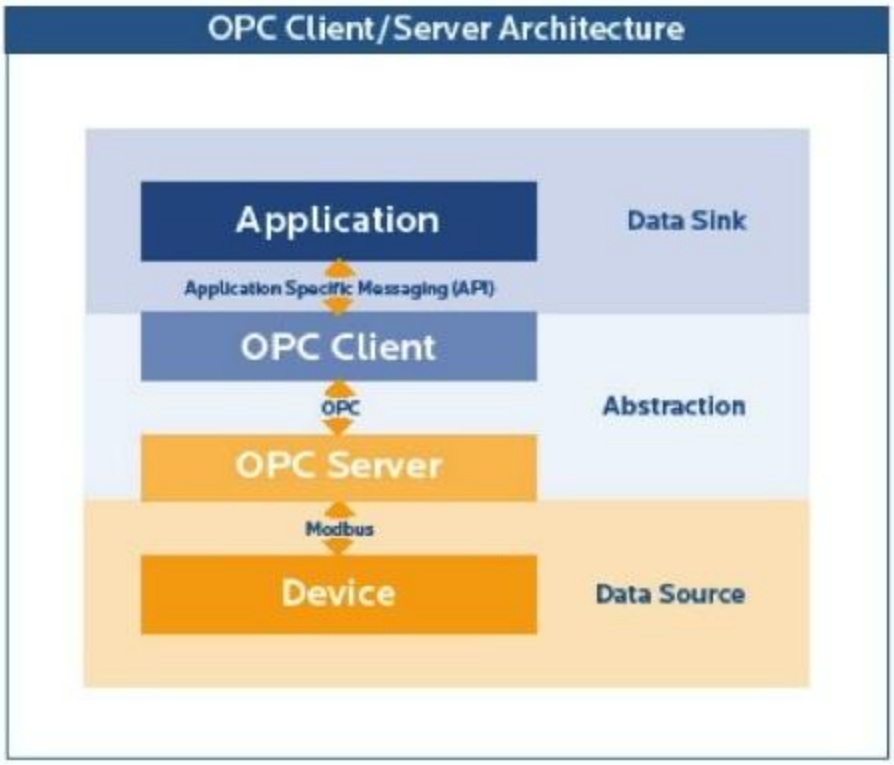

Figura #3: Una mejor vista del funcionamiento OPC nos revela dos componentes: El Cliente OPC y el Servidor OPC. La especificación OPC define el mensaje entre estos dos componentes.

1.3.1.3 Servidores y clientes OPC

Los servidores OPC son conectores que se pueden asimilar a traductores entre el mundo OPC y los protocolos nativos de una Fuente de Datos. OPC es bidireccional, esto es, los Servidores OPC pueden leer de y escribir en una Fuente de Datos. La relación Servidor OPC/Cliente OPC es de tipo maestro/esclavo, lo que significa que un servidor OPC solo transferirá datos de/a una Fuente de Datos si un Cliente OPC así lo pide.

Un Cliente OPC representa un destino de datos. Inician y controlan la comunicación con Servidores OPC basados en las peticiones recibidas desde la aplicación en la que están embebidos. Los Clientes OPC traducen las peticiones de comunicación provenientes de una aplicación dada de la petición OPC equivalente y la envía al Servidor OPC adecuado para que la procese. A cambio, cuando los datos OPC vuelven del Servidor OPC, el Cliente OPC los traduce al formato nativo de la aplicación para que ésta pueda trabajar de forma adecuada con los datos. Técnicamente los Clientes OPC son módulos de software utilizados por una aplicación para permitirle comunicarse con cualquier Servidor OPC compatible visible en la red.

Los proveedores de servidores OPC como aplicaciones finales generalmente implementan solo un protocolo, por lo que para cubrir una amplia gama de protocolos se

deben adquirir esta misma cantidad de servidores por parte de los clientes. Otros proveedores de instrumentación de campo solo desarrollan servidores OPC para su hardware específico y en muchas ocasiones no se conocen los protocolos que implementan.

En realidad OPC es un conjunto de numerosos protocolos entre los que destacamos los siguientes:

- **OPC-DA (Data Access):** El original, sirve para el intercambio de datos a tiempo real entre servidores y clientes.

La parte de acceso a datos ha sido diseñada para proporcionar a los usuarios los medios para integrar hardware y software. Se trata de un mecanismo de comunicación cliente / servidor. Un servidor de acceso a datos accederá normalmente a los dispositivos que utilizan los drivers y / o protocolos adecuados y permitir que las aplicaciones clientes de acceso a datos puedan acceder a estos dispositivos de una manera uniforme.

- **OPC-AE (Alarms & Events):** Proporciona alarmas y notificaciones de eventos.

Este sistema es un complemento de las otras interfaces OPC (sobre todo de acceso a datos). Las alarmas y los eventos incluyen las alarmas de proceso, operador de solicitudes de acciones manuales generados por el sistema, mensajes informativos, completamiento de lote, rastreo y auditorias de mensajes tales como un cambio en un parámetro de referencia por un operador.

- **OPC HDA (Historical Data Access):** Acceso histórico a datos OPC.

Proporcionar un conjunto de interfaces estándar que permitirá a los clientes acceder a los archivos históricos para recuperar y almacenar datos de manera uniforme. La intención es la de permitir el acceso a una amplia gama de archivos de datos, desde un sistema simple de registro de datos en serie a un sistema SCADA complejo, con métodos de interfaz que permitan la presentación y manipulación de los datos en las formas más comunes en la industria, tales como tendencias, actualización automática de los nuevos valores, la representación gráfica de los procesos de negocio y los datos agregados calculados.

Se seleccionó el protocolo de Acceso a Datos puesto que es el protocolo OPC rector en el intercambio de datos en tiempo real, lo cual es una de las principales pautas a medir en cuanto a la selección, por tanto le da solución a la problemática planteada y suple la necesidad de un protocolo en la conexión entre los dispositivos físicos y la herramienta

implementada.

1.3.1.4 Desarrollo de OPC en el mundo y Cuba

Se han desarrollado un gran número de aplicaciones que utilizan el estándar OPC, con el objetivo de aprovechar al máximo los recursos tecnológicos de hoy en día. A continuación se mencionan algunas aplicaciones que hacen uso de la Tecnología OPC:

En España, en una empresa energética se utiliza la tecnología OPC para el funcionamiento de sus parques eólicos. Esta empresa energética posee una aplicación integrada con OPC DA y HDA. Cuenta con 650.000 puntos de distribución de energía/clientes particulares, y proporciona acceso a datos centralizado para tres parques eólicos remotos situados en España. Los parques eólicos contienen turbinas y sistemas de control de diferentes fabricantes, con diversos medios de comunicación y con un bajo ancho de banda para tener un acceso a la información en tiempo real de los equipos de vigilancia, así como la toma de decisiones de producción (MatrikonOPC, 2013). Esta aplicación sólo utiliza al OPC como un recolector de datos y no en un sentido bidireccional en el que se le puedan introducir datos y modificarlos haciendo así un proceso cambiante. Por estas razones no dan solución al problema identificado.

Cray Valley es uno de los líderes mundiales en la fabricación y comercialización de resinas. Su actividad abarca tres líneas de producto: resinas y aditivos para recubrimientos, resinas para materiales compuestos y resinas de hidrocarburo y aditivos funcionales. Operan con una plataforma que integra los diversos sistemas de control que existen en cada una de las plantas. Además permite gestionar eficientemente las diversas áreas del negocio. La integración de estos sistemas de control permite a la empresa, la optimización de la gestión empresarial y las operaciones de las plantas en tiempo real. Para ello utilizaron los protocolos OPC DA y AE. Esta plataforma tampoco resuelve el problema planteado porque se utiliza como una aplicación de alarmas y eventos para el aviso constante de los puntos críticos en las plantas.

Se conoce del desarrollo de Sistemas de Control Industrial (ICS) en Cuba, que en el año 1991 fue implementado por ingenieros de la Unión del Níquel en la provincia de Holguín un ICS conocido como Sistema de Supervisión y Control de Procesos EROS. Este se usó en 27 plantas de diferentes tipos en el país. Puede trabajar acoplado a diversos sistemas de colección de datos, como elemento único o formando parte de una red industrial. Esta herramienta se conoce como el primer SCADA realizado en el país, el cual en sus inicios

resultó ser un importante avance en el desarrollo de este tipo de tecnología. Sin embargo fue quedando obsoleto por lo que fue descartado en la selección.

En 2009 en la Universidad de la Ciencias Informáticas (UCI) se desarrolló una integración OPC, al sistema "SCADA Guardián del ALBA", por la existencia de un elevado número de dispositivos de campo propietarios en las instalaciones de PDVSA, permitiéndole incorporarlos en las implantaciones del sistema. La integración OPC al Guardián del ALBA es una es una de las herramientas en las cual el grupo de desarrollo se apoyó para la realización del proyecto. Este proyecto se confeccionó con ideas tomadas del sistema EROS. Sin embargo no es factible la utilización del mismo puesto que utilizan TCP/IP en la conexión, propiciando que la caída del sistema produzca graves errores ya que el sistema es en tiempo real.

A partir del estudio realizado se puede concluir que ninguna de los sistemas encontrados dan respuesta al problema plantado, por lo que se hace necesario el desarrollo del estándar OPC adecuado a las características del NVP.

1.4 Metodología de desarrollo

La determinación de una metodología de desarrollo del software, es un factor determinante en el éxito de un proyecto. Surge ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto de software. Su selección incluye un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo de software. Existen dos grandes grupos de metodologías:

- **Las metodologías tradicionales:** Este tipo de metodología requiere de una extensa documentación durante todo el ciclo de vida, ya que pretende prever todo de antemano. Suelen ser eficaces y necesarias cuando se trata de proyectos que requiere de grandes equipos de desarrollo. Dentro de estas metodologías una de las más utilizadas es la Metodología RUP (Rational Unified Process), la cual divide el desarrollo en cuatro fases que definen su ciclo de vida y en nueve flujos de trabajo, seis de ingeniería y tres de soporte (Jacobson, 2000).

  Las principales características de RUP son:

  - ✓ Centrado en la arquitectura: la arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por plataformas de

software, sistemas operativos, manejadores de bases de datos, consideraciones de desarrollo como sistemas heredados y requerimientos no funcionales. Una vez definida la arquitectura se puede decir que el sistema tiene forma.

- ✓ Dirigido por casos de uso: tiene a los casos de uso como el hilo conductor del proceso de desarrollo, los desarrolladores crean una serie de modelos de diseño e implementación que los llevan a cabo. Además, estos modelos se validan para que sean conformes a los casos de uso. Finalmente, los casos de uso también sirven para realizar las pruebas sobre los componentes desarrollados.
- ✓ Iterativo e incremental: esta característica propone dividir el proceso de desarrollo en partes, cada una de las cuales incluya las fases de: Requerimientos, Análisis, Diseño, Implementación y Pruebas, con el objetivo de acelerar el ritmo de desarrollo para que el producto salga al mercado en el menor tiempo posible y con mayor calidad.

RUP no es factible para el cliente OPC del NVP, debido a que el sistema por sus características técnicas y se hace muy difícil su descripción en casos de uso. No obstante por su complejidad se necesita una amplia documentación por lo que se elaborarán los artefactos diagramas de despliegue y componentes que propone RUP.

- **Las metodologías ágiles:** Para este tipo de metodologías es más importante la capacidad de respuesta ante los cambios realizados que el seguimiento estricto de un plan. Se enfatiza en la satisfacción del cliente y promueve el trabajo en equipo. Una de las metodologías más utilizadas es la Metodología XP (Extreme Programming) (Wells. 2009) la cual se basa en una serie de valores, principios y prácticas que brindan una satisfactoria productividad en el proceso de desarrollo del software.

  Durante el ciclo de vida en dicha metodología aparecen cambios frecuentes, por lo que a veces el equipo que lo integra no se encuentra preparado para enfrentarlos, ante tal situación el equipo de desarrollo enfrenta un conjunto de valores que son importantes para el logro del producto, entre los que se encuentran:

  1. Comunicación

2. Coraje.
3. Simplicidad.
4. Retroalimentación.

Cuenta también con los siguientes principios y artefactos, los cuales son indispensables en el desarrollo de la metodología:

- ✓ Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que se adelanta en algo hacia el futuro, se puede hacer pruebas de las fallas que pudieran ocurrir. Es como si se adelantara a obtener los posibles errores.

- ✓ Re fabricación: se desarrolla en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

- ✓ Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

Artefactos esenciales en XP:

1. Historias del usuario.
2. Pruebas de aceptación.
3. Metáfora.
4. Tareas de ingeniería.
5. Pruebas unitarias y de integración.
6. Plan de iteraciones.
7. Código.

Luego de haberse realizado el estudio de las metodologías, se decidió centrar el desarrollo del producto sobre la metodología ágil, Extreme Programming (XP) teniendo en cuenta que el grupo de desarrollo encargado del producto de software es muy reducido. Otras de las características que conlleva a la selección de esta metodología son el corto plazo de sus iteraciones y la constante presencia de un cliente, lo cual provoca un constante cambio, reflejándose por concerniente la flexibilidad y la alta respuesta a dichos cambios en el menor tiempo posible.

Además de seleccionar la metodología XP, la cual se propone para guiar el transcurso del desarrollo del software, el equipo de desarrollo propone el uso necesario de algunos artefactos de la metodología RUP que propiciarán un mayor entendimiento a los usuarios, referente al flujo de la aplicación y sus principales componentes.

1.4.1 Descripción de los artefactos a utilizar

Para estructurar, planificar y controlar el proceso de desarrollo de software del cliente OPC para el NVP se proponen los siguientes tipos de artefactos:

- ✓ **Historias del usuario**: son la técnica utilizada en XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales.
- ✓ **Plan de iteraciones:** define exactamente cuáles historias de usuario serán implementadas en cada iteración. Se toma como base cada una de las historias de usuarios y el esfuerzo que se requiere para el desarrollo de estas, y se procede a fragmentar el trabajo en iteraciones.
- ✓ **Metáfora:** es un pequeño resumen sobre cómo debe funcionar el componente con el objetivo de que el dominio del problema sea fácilmente comprendido. La selección de una metáfora permite mantener la coherencia e integridad conceptual de todos los elementos a implementar.
- ✓ **Tarjeta CRC:** representan objetos, para los cuales se especifican la clase a la que pertenece dicho objeto, las responsabilidades u objetivos que debe cumplir y las clases que colaboran con cada responsabilidad.
- ✓ **Tareas de ingeniería:** Las tareas de ingeniería son un conjunto de acciones a desarrollar para resolver las historias de usuario. Permiten organizar el proceso de implementación además de posibilitar que sea conocido el grado de complejidad de cada historia de usuario teniendo en cuenta la cantidad de tareas asociadas a ella.
- ✓ **Pruebas unitarias:** constituyen una forma de probar el correcto funcionamiento de un módulo de códigos, por lo que también se le conoce como pruebas modulares. Tienen como objetivo asegurar que cada uno de los módulos de códigos funcione correctamente por separado.

- ✓ **Pruebas de aceptación:** constituyen pruebas basadas en la ejecución, revisión y retroalimentación de las funcionalidades que han sido diseñadas para el software. Estas pruebas se realizan a través de modelos de prueba conocidos como casos de prueba
- ✓ **Diagrama de Clases del Diseño:** es un tipo de diagrama estático que describe la estructura de un sistema y las relaciones entre los principales elementos, dígase clases e interfaces y las relaciones que existe entre ellos. El diagrama de clases le permite al cliente tener una buena perspectiva de cómo está estructurado el diseño de la herramienta, puesto que en XP, con el uso de las tarjetas CRC, no es apreciable de una forma visual las distintas interconexiones entre las clases, sino que se realiza por medio de tablas.

- ✓ **Diagrama de componente**: 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 la aplicación informática. El diagrama de componentes es uno de los diagramas no generado por la metodología XP. Permite al usuario obtener una visión de los principales componentes y en qué capa están presentes en la arquitectura. Además se puede apreciar cómo está realizada la integración del estándar OPC al NVP.

- ✓ **Diagrama 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. El diagrama de despliegue es uno de los diagramas no generado por la metodología XP. Permite al usuario ver de una forma general los principales componentes por los que está compuesta la aplicación.

1.5 Herramientas y tecnologías seleccionadas

Con el propósito de desarrollar un componente con la mayor calidad posible y que favorezca la integración con el NVP, se decidió mantener las tecnologías y herramientas seleccionadas por el equipo de desarrollo del NVP. Estas son:

- Lenguaje de modelado, UML 8.0.
- Herramienta de modelado, Visual Pradigm 5.0.

- Lenguaje de programación, C++.
- Entorno de desarrollo (IDE), Qt-Creator.

1.5.1 Lenguaje de Modelado Unificado (UML 8.0)

"UML es un lenguaje más expresivo, claro y uniforme 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).

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. Pueden ser artefactos: un modelo, una descripción que comprende el desarrollo de software que se base en el enfoque Orientado a Objetos. Además es el lenguaje de modelado que se utiliza en la herramienta Visual Paradigm la cual es la que utiliza el equipo de desarrollo para la modelación de sus clases.

1.5.2 Visual Paradigm para UML 8.0 Enterprise Edition

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.

Entre las herramientas CASE orientadas a UML se pueden encontrar Visual Paradigm, la cual fue seleccionada en tesis anteriores por sus diversas ventajas. (Ortiz, 2008), (Cleger, 2009), (García, 2011).

1.5.3 Lenguaje de programación C++

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 bibliotecas 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# y entre los puntos que se tuvieron en cuenta para seleccionar la herramienta se encuentran:

Portabilidad.
Capacidades 2D/3D.
Matemáticas de precisión compleja.
Gestión de memoria.
Velocidad de ejecución.
Licencia.
Eficiencia.
Modularidad.

Después de terminar el estudio se ponderaron cada uno de estos puntos según el peso que tenían para el NVP para cada lenguaje, y se obtuvo como resultado C++ acumuló 48 puntos, C# 28 puntos y Java 39 puntos. Teniendo en cuenta estos resultados se optó por escoger como lenguaje de programación C++ para aprovechar su velocidad de ejecución y gestión de memoria además de todas las potencialidades que ofrece de manera general. (Tomado de las tesis (Ortiz, 2008), (Cleger, 2009), (García, 2011)).

1.5.4 Entorno de desarrollo integrado (IDE) Qt-Creator

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 las más utilizadas se encuentran KDevelop, Eclipse y Qt-Creator. Se seleccionó Qt-Creator debido a que simplifica el desarrollo de aplicaciones desktop para programadores C++. 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. Tomado de las tesis (Ortiz, 2008), (Cleger, 2009), (García, 2011).

1.6 Métodos y técnicas de pruebas de software

Las pruebas constituyen un elemento fundamental para asegurar la calidad de los diversos software. Debido a que cada día el desarrollo del software resulta más complejo y las posibilidades de cometer errores aumentan, es necesario llevar a cabo numerosas actividades para erradicar dichos errores y comprobar la efectividad de las operaciones utilizadas, para de esta forma satisfacer las necesidades del cliente.

Para medir la calidad del análisis y el diseño se proponen un conjunto de métricas que permitirán medir la calidad de la especificación de los requisitos, así como evaluar la calidad de los atributos internos del sistema.

XP divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores, y pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñada por el cliente final.

- ✓ **Pruebas unitarias (caja blanca)**: Son pruebas estructurales, que conociendo el código y siguiendo su estructura lógica, se pueden diseñar pruebas destinadas a comprobar que el código hace correctamente lo que el diseño de bajo nivel indica. Ejemplos típicos de ello son las pruebas unitarias las cuales se centran en lo que hay codificado o diseñado a bajo nivel por lo que no es necesario conocer la especificación de requisitos.

- ✓ **Pruebas funcionales (caja negra):** Son pruebas funcionales, que se realizan sobre la interfaz empleando un determinado conjunto de datos de entrada y observando las salidas que se producen para determinar si la función se está desempeñando correctamente por el sistema bajo prueba No es necesario conocer la lógica del programa, únicamente la funcionalidad que debe realizar.

1.6.1 Validación del análisis

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 proponen un total de cuatro métricas: especificidad (ausencia de ambigüedad), compleción, corrección, comprensión, y la capacidad de verificación. (Overmyer, 1993)

- ✓ **Especificidad:** se utiliza 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.

- ✓ **Compleción:** se utiliza para determinar cuan completos se encuentran los requisitos, evitando la pobre especificación.

- ✓ **Corrección:** una especificación se considera correcta cuando cada requisito contenido en ella represente una característica que el sistema debe poseer.

- ✓ **Comprensión:** La comprensión de los requisitos se determina cuan comprensible se encuentran los requisitos especificados.

1.6.2 Validación del diseño

Las métricas del software son una medida cuantitativa de evaluar la calidad de los atributos internos de un sistema. Se emplean con el objetivo de llevar el control de la calidad del producto que se está desarrollando, evaluar la efectividad del proceso y mejorar la calidad del trabajo. Las métricas proporcionan los conocimientos necesarios para crear modelos efectivos de análisis y diseño, un código sólido y pruebas exhaustivas. (Pressman R. S., 2005)

Para la realización de estas pruebas se realizarán las métricas Tamaño Operacional de las Clases (TOC) y Relaciones entre Clases (RC).

- ✓ **Tamaño Operacional de Clases:** consiste en un cálculo de los atributos u operaciones totales que se realizan en las clases, esta métrica mide el grado de responsabilidad, complejidad y reutilización que poseen las clases.

- ✓ **Relaciones entre Clases:** está dada por el número de relaciones de uso de una clase con otras. La aplicación de dicha métrica permite evaluar atributos como el Acoplamiento, la Complejidad de mantenimiento, la Reutilización y la Cantidad de

pruebas.

1.6.3 Pruebas unitarias (caja blanca)

Las pruebas unitarias, conocidas como prueba de caja blanca, constituyen una forma de probar el correcto funcionamiento de un módulo de códigos, por lo que también se le conoce como pruebas modulares. Tienen como objetivo asegurar que cada uno de los módulos de códigos funcione correctamente por separado. Estas pruebas proporcionan cuatro ventajas básicas:

- ✓ Fomentan el cambio: Estas pruebas facilitan que el programador modifique el código para mejorar su estructura, por lo que se vuelven a hacer pruebas sobre los cambios realizados, para asegurarse de que los nuevos cambios no han introducido errores.

- ✓ Documentan el código: Las propias pruebas constituyen documentación del código puesto que en este se puede ver cómo utilizarlo.

- ✓ Separan la interfaz y la implementación: Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas. Se puede cambiar cualquiera de los dos sin afectar al otro.

- ✓ Los errores están más acotados y son más fáciles de localizar.

Se propone el desarrollo de las pruebas unitarias para la validación de la codificación, pues permiten comprobar que la aplicación tenga el rendimiento deseado con la mayor exactitud y velocidad posibles, parámetros que son indispensables en una herramienta en tiempo real. Para ello se utilizará la técnica del camino básico que permite definir los diferentes caminos independientes de la función, y probar su funcionamiento al menos una vez.

1.6.4 Pruebas funcionales (caja negra)

Las pruebas funcionales, conocidas como pruebas de caja negra, constituyen pruebas basadas en la ejecución, revisión y retroalimentación de las funcionalidades que han sido diseñadas para el software. Estas pruebas se realizan a través de modelos de prueba

conocidos como casos de prueba, que buscan evaluar cada una de las opciones con la que cuenta el sistema informático, enfocando su atención a las respuestas del sistema de acuerdo a los datos de entrada y su resultado en los datos de salida.

A través de estas pruebas se valida si los requisitos que fueron definidos por el cliente cumplen su funcionalidad de forma adecuada en el sistema, buscando de esta forma reducir fallos en el sistema y permitiendo la corrección en etapas tempranas.

En el presente trabajo se propone el desarrollo de las pruebas funcionales para la validación del sistema informático, ya que estas comprueban los requisitos funcionales que debe cumplir dicho sistema, asegurando de esta forma su correcto funcionamiento. Para ello se realizará el diseño de los casos de pruebas por medio de la técnica de partición de equivalencia, describiéndose las entradas que se realizarán al sistema y las salidas que generará el mismo.

1.7 Conclusiones Parciales

En el presente capítulo se sistematizaron los referentes teóricos relacionados con la simulación de procesos industriales y los nodos virtuales, y se estableció la necesidad de incorporar a la herramienta Nodo Virtual de Procesos (NVP), un estándar de comunicación que permita la incorporación de controladores físicos.

Se realizó un estudio de los principales estándares de comunicación existentes y se optó por un cliente OPC ya que es un estándar robusto, fiable que integra otros protocolos como DCOM, TCP/IP y DDE. Permite que el sistema pueda comunicarse con una mayor diversidad de dispositivos. Además se realizó un estudio de los sistemas existentes que implementan clientes OPC para la solución de problemas similares, y se concluyó que no satisfacían los requerimientos del problema planteado por lo que debía desarrollarse un cliente OPC para el NVP.

Se seleccionó la metodología de desarrollo Extreme Programing (XP) con la incorporación de los artefactos de RUP diagrama de clases, de componentes y de despliegue. Para ello se tuvo en cuenta que el sistema era muy difícil representarlo a través de casos de uso, que el grupo de desarrollo encargado del producto de software es muy reducido, así como el corto plazo de sus iteraciones y la constante presencia de un cliente, que provoca un constante cambio, reflejándose por concerniente la flexibilidad y la alta respuesta a dichos

cambios en el menor tiempo posible.

Como herramientas y tecnologías para el análisis, diseño e implementación se mantienen las seleccionadas por el equipo de desarrollo del NVP para propiciar la integración y el correcto funcionamiento del sistema. Estas son Visual Paradigm 5.0 como herramienta de modelado, UML 8.0 como lenguaje de modelado. Además se seleccionó como lenguaje de programación C++ y como IDE de desarrollo Qt-Creator 2010.

Por último se definieron para la validación del sistema las métricas especificidad, compleción, corrección y comprensión en el análisis, y tamaño operacional de clases y relación entre clases en el diseño, además pruebas de caja blanca como camino básico y de caja negra como casos de pruebas. Dejando todo listo para comenzar la planificación, el diseño y la implementación.

CAPÍTULO 2: PLANIFICACIÓN, DISEÑO E IMPLEMENTACIÓN

Introducción

En este capítulo se realiza una descripción de las tres primeras fases que presenta la metodología XP (Planificación, Diseño y Desarrollo) y los principales artefactos en estas fases. Entre los artefactos que se generarán se encuentran las historias de usuarios, el plan de iteraciones, las tareas de ingeniería y las tarjetas CRC, obteniendo de esta forma un producto con la calidad requerida por el usuario final, siendo este el principal objetivo a alcanzar.

2.1 Planificación

La planificación en XP está basada en un conjunto de decisiones tomadas por el cliente de conjunto con los programadores. Los clientes representan las necesidades propias del negocio como son la prioridad de las historias de usuario y las fechas de entrega de cada versión del producto final. Mientras que los programadores definen los requerimientos técnicos que complementan las necesidades del negocio, como por ejemplo la duración estimada de la implementación de las historias de usuario y la organización del proceso de desarrollo en general.

2.1.1 Requerimientos del sistema

Los requisitos funcionales son capacidades o cualidades que el sistema debe tener para cubrir las necesidades de los usuarios. En cambio, los requisitos no funcionales van a constituir las propiedades o cualidades que el producto debe tener para que llegue a ser atractivo, confiable, usable y rápido. De acuerdo a las clasificaciones y descripciones antes mencionadas se han definido los siguientes requerimientos.

Requisitos funcionales del sistema.

HU1: Crear Host

- RF1: Crear Host

HU2: Conectar al cliente OPC

- RF2: Conectar al cliente OPC

HU3: Gestionar Grupo

- RF3: Crear Grupo
- RF4: Habilitar/Deshabilitar grupo
- RF5: Eliminar grupo

HU4: Gestionar Ítem

- RF6: Crear Ítem
- RF7: Modificar Ítem
- RF8: Eliminar Ítem

HU5: Conectar al servidor NVP

- RF9: Conectar al servidor NVP

HU6: Mostrar árbol de herencia

- RF10: Mostrar árbol de herencia

Requisitos no funcionales del sistema.

✓ Requerimientos de Software:
- Plataforma .NET, Windows XP o más avanzado,

✓ Requerimientos de Hardware:
- Microprocesador P4, Memoria RAM de 2GB, 1GB de espacio en disco o más.

✓ Requerimientos en el diseño y la implementación:
- Lenguaje de programación: C++
- Entorno de desarrollo: Qt-Creator.

✓ Requerimientos de Seguridad:
- El administrador es el único que tiene el control total del sistema.
- Para realizar una operación determinada el usuario deberá estar autenticado y podrán acceder al mismo de acuerdo a los roles asignados.
- Los cambios en el sistema sólo pueden ser realizados por los usuarios autorizados.
- El sistema podrá ser usado en cualquier momento por todos los usuarios autorizados.

✓ Apariencia o interfaz externa:

- El sistema tiene que ofrecer una interfaz amigable, fácil de operar.
- El sistema tiene que mantener la línea de diseño establecida para la institución que mantiene la uniformidad y representatividad de la misma.
- Las interfaces tienen que mostrar un diseño sencillo, con pocas entradas, permitiendo un balance adecuado entre funcionalidad y simplicidad de tal manera que no se haga difícil para los usuarios la utilización del sistema.

✓ Usabilidad:

- El software tendrá siempre la posibilidad de ayuda disponible para cualquier tipo de usuario, lo que le permitirá un avance considerable en la explotación de la aplicación en todas sus funcionalidades.
- Tiene que poseer una interfaz agradable para el cliente.

2.1.2 Historias de usuario

Las historias de usuario (HU) son la técnica utilizada en XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales.

El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento historias de usuario pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (Jeffries, 2001).

Respecto de la información contenida en la historia de usuario, existen varias plantillas sugeridas pero no existe un consenso al respecto. En muchos casos sólo se propone utilizar un nombre y una descripción (Wake, 2002) o sólo una descripción (Jeffries, 2001), más quizás una estimación de esfuerzo en días (Newkirk**,** 2001). Beck en su libro (Beck, K. 2000) presenta un ejemplo de ficha (customer story and task card) en la cual pueden reconocerse los siguientes contenidos: fecha, tipo de actividad (nueva, corrección, mejora), prueba funcional, número de historia, prioridad técnica y del cliente, referencia a otra historia previa, riesgo, estimación técnica, descripción, notas y una lista de seguimiento con la fecha, estado cosas por terminar y comentarios.

Las historias de usuario deben ser:

Valoradas por los clientes o usuarios: Los intereses de los clientes y de los usuarios no siempre coinciden, pero en todo caso, cada historia debe ser importante para alguno de ellos más que para el desarrollador.

Estimables: Un resultado de la discusión de una historia de usuario es la estimación del tiempo que tomará completarla. Esto permite estimar el tiempo total del proyecto.

Pequeñas: Las historias muy largas son difíciles de estimar e imponen restricciones sobre la planificación de un desarrollo iterativo. Generalmente se recomienda la consolidación de historias muy cortas en una sola historia.

Verificables: Las historias de usuario cubren requerimientos funcionales, por lo que generalmente son verificables. Cuando sea posible, la verificación debe automatizarse, de manera que pueda ser verificada en cada entrega del proyecto.

Durante la fase de exploración se identificaron seis Historias de Usuarios (HU) representando cada una las funcionalidades del sistema, las cuales se describen a continuación (Ver Tablas de la 1 a la 6):

Historia de Usuario	
Número: 1	**Nombre de la Historia de Usuario:** Crear host.
Cantidad de modificaciones a la Historia de Usuario: Ninguna.	
Usuario: Adrián Sánchez Parra.	**Iteración asignada:** 1.
Prioridad en negocio: Alta.	**Puntos estimados:** 1 semana.
Riesgo en desarrollo: Bajo.	**Puntos reales:** 1 semana.
Descripción: El sistema debe permitir al usuario crear un host el cual puede ser localhost o una dirección IP fija en la cual se encuentra el servidor del Nodo Virtual de Procesos.	
Observaciones:	
Prototipo de interfaces:	

Tabla 1: HU Crear Host

Historia de Usuario	
Número: 2.	**Nombre de la Historia de Usuario:** Conectar al servidor OPC.
Cantidad de modificaciones a la Historia de Usuario: Ninguna.	
Usuario: Adrián Sánchez Parra.	**Iteración asignada:** 1.
Prioridad en negocio: Alta.	**Puntos estimados:** 2 semanas.
Riesgo en desarrollo: Medio.	**Puntos reales:** 2 semanas.
Descripción: El sistema muestra los servidores disponibles para que el usuario seleccione con cual quiere establecer la conexión y se conecta. Puede ser habilitado o deshabilitado.	
Observaciones:	
Prototipo de interfaces:	

Tabla 2: HU Conectar al Servidor OPC.

Historia de Usuario	
Número: 3.	**Nombre de la Historia de Usuario:** Gestionar Grupo.
Cantidad de modificaciones a la Historia de Usuario: 2.	
Usuario: Francisco Rubio Santiesteban.	**Iteración asignada:** 1.
Prioridad en negocio: Alta.	**Puntos estimados:** 3 semanas.
Riesgo en desarrollo: Alto.	**Puntos reales:** 3 semanas.
Descripción: El sistema debe adicionar, modificar y eliminar un grupo el cual ya debe tener asignado un servidor, siendo los grupos el contenedor de los Ítem. Puede ser habilitado o deshabilitado.	

Observaciones:	
Prototipo de interfaces:	

Tabla 3: HU Gestionar Grupo.

Historia de Usuario	
Número: 4.	**Nombre de la Historia de Usuario:** Gestionar Ítem.
Cantidad de modificaciones a la Historia de Usuario: Ninguna.	
Usuario: Francisco Rubio Santiesteban.	**Iteración asignada:** 2.
Prioridad en negocio: Alta.	**Puntos estimados:** 3 semanas.
Riesgo en desarrollo: Alto.	**Puntos reales:** 3 semanas.
Descripción El sistema debe adicionar, modificar y eliminar un Ítem el cual ya debe tener asignado un grupo correspondiente, además se pude seleccionar de modo solo lectura o lectura/escritura.	
Observaciones:	
Prototipo de interfaces:	

Tabla 4: HU Gestionar Ítem.

istoria de Usuario	
Número: 5.	**Nombre de la Historia de Usuario:** Conectar al servidor NVP.
Cantidad de modificaciones a la Historia de Usuario: 1.	
Usuario: Adrián Sánchez Parra.	**Iteración asignada:** 1.
Prioridad en negocio: Media.	**Puntos estimados:** 2 semanas.
Riesgo en desarrollo: Medio.	**Puntos reales:** 2 semanas.

Descripción: El sistema debe conectarse al servidor NVP y enviar y recibir datos.
Observaciones:
Prototipo de interfaces:

Tabla 5: HU Conectar al servidor NVP.

Historia de Usuario	
Número: 6.	**Nombre de la Historia de Usuario:** Mostrar árbol de herencia.
Cantidad de modificaciones a la Historia de Usuario: Ninguna.	
Usuario:	**Iteración asignada:** 2.
Prioridad en negocio: Baja.	**Puntos estimados:** 3 semanas.
Riesgo en desarrollo: Alta.	**Puntos reales:** 3 semanas.
Descripción: El sistema debe mostrar el árbol de conexión de todos los nodos participantes dígase: Servidores, Grupos, Ítems y Host.	
Observaciones:	
Prototipo de interfaces:	

Tabla 6: HU Mostrar árbol de herencia.

2.1.3 Estimación de esfuerzos por historias de usuario

Se realizó una estimación de esfuerzo que definirá el tiempo que costará desarrollar cada una de las historias de usuario identificadas, para solución del sistema propuesto, llegando a los resultados que se muestran a continuación en la Tabla 7:

Historia de Usuario	Punto de estimación (semanas)
Crear host.	1
Conectar al cliente OPC.	2

Gestionar Grupos.	3
Gestionar Ítem.	3
Conectar al servidor NVP.	2
Mostrar árbol de herencia.	3

Tabla 7: Estimación de esfuerzos por historias de usuario

2.1.4 Plan de iteraciones

Este plan incluye varias iteraciones sobre el sistema antes de ser entregado. Los elementos que deben tenerse en cuenta durante su elaboración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas y tareas no terminadas en la iteración. Este plan define exactamente cuáles historias de usuario serán implementadas en cada iteración. Tomando como base cada una de las historias de usuarios y el esfuerzo que se requiere para el desarrollo de estas, se procede a fragmentar el trabajo en dos iteraciones; obteniendo un trabajo incremental, donde la piedra angular es la comunicación entre el equipo de desarrollo y el cliente.

A continuación en la Tabla 8 se describen cada una de las iteraciones propuestas donde la duración total de iteraciones en días se obtiene a partir del esfuerzo en días estimado por el desarrollador para implementar cada historia de usuario:

Reléase	Descripción de la iteración	Orden de la HU a implementar
1	Se tuvieron en cuenta aquellas HU de mayor importancia en cuanto a la funcionalidad que describen cada una, que se ve reflejado en la alta prioridad de negocio.	▪ Conectar al cliente OPC. ▪ Gestionar Grupo. ▪ Gestionar Ítem. ▪ Crear host.

2	Se tuvieron en cuenta aquellas funcionalidades del sistema con menos prioridad, es decir, aquellas HU que presentan prioridad media o baja. Por otra parte en esta iteración se corregirán los errores encontrados en la iteración anterior, obteniéndose una nueva versión del sistema.	▪ Conectar al servidor NVP. ▪ Mostrar árbol de herencia.

Tabla 8: Plan de Iteraciones.

2.1.5 Plan de duración de las iteraciones

Este plan tiene como objetivo reflejar detalladamente las HU que se desarrollarán en cada iteración, la duración de cada una, así como el orden en que serán implementadas en dichas iteraciones. En este caso se crea un solo plan de iteraciones debido a que existe un único equipo de desarrolladores (Tabla 9).

Nro. de iteración.	Historia de usuario a implementar.	Duración total de la iteración (semanas).
1	Crear host. Conectar al cliente OPC. Gestionar Grupo. Gestionar Ítem.	9
2	Conectar al servidor NVP. Mostrar árbol de herencia.	5

Tabla 9: Plan de duración de las iteraciones.

2.1.6 Plan de entregas

El plan de entregas no es más que un cronograma con las HU que serán implementadas en cada iteración, es decir, un cronograma con las entregas de partes funcionales del

software. A continuación se hace una propuesta de la fecha aproximada en que se harán versiones al sistema al finalizar cada iteración en la fase de implementación:

Ya elaboradas por el cliente las distintas historias de usuarios y confeccionado por los desarrolladores el plan de iteración se procede a la confección del plan de entrega con la intención de que los mismos obtengan una estimación real, fijándose un periodo de tiempo sobre el cual se debe de implementar cada historia de usuario y definiéndose el grado de dificultad de la misma.

A continuación en la Tabla 10 se presenta el plan de entrega elaborado por el equipo de desarrollo donde se reflejan las fechas de entregas para las primeras versiones de las historias de usuario:

Número de iteración.	Historia de usuario a implementar.	Fecha de entrega.
1	Crear host. Conectar al cliente OPC. Gestionar Grupo. Gestionar Item.	11/03/ 2013
2	Conectar al servidor NVP. Mostrar árbol de herencia.	15/04/2013

Tabla 10: Plan de Entrega.

2.2 Diseño

La metodología de desarrollo de software XP sugiere que se realicen diseños simples y sencillos, para de esta forma lograr que sean de fácil entendimiento en la fase de implementación, lo que le costará menos tiempo al desarrollador llevar a cabo esta tarea. En esta fase el sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo, la cual describe como debería funcionar el sistema, a través del diseño de las tarjetas CRC (Class, Responsibilities and Collaboration por sus siglas en inglés).

2.2.1 Metáfora

Una metáfora es un pequeño resumen sobre cómo debe funcionar el componente con el objetivo de que el dominio del problema sea fácilmente comprendido. La selección de una metáfora permite mantener la coherencia e integridad conceptual de todos los elementos a implementar. A continuación se define la metáfora global que guía el desarrollo del presente proyecto.

El sistema propuesto se basa en el desarrollo de un cliente OPC para dar solución al problema planteado en el capítulo anterior, que no es más que añadir en el funcionamiento del NVP los controladores físicos. Para ello desde el servidor del Nodo Virtual, se le realizarán peticiones al cliente OPC. El Cliente OPC seleccionará el servidor al cual está dirigida la petición y se le asignará a dicho servidor. Los servidores tendrán un conjunto de grupos y estos a su vez contendrán a los Ítems, que representan conexiones a las fuentes de datos (Dispositivos físicos). Es bueno señalar que un Ítem, no es accesible como un objeto por parte de un cliente OPC. Por lo tanto, no hay ninguna interfaz externa definida para un Ítem. Todos los accesos a Ítems se hacen a través de un objeto del Grupo que contiene al Ítem. Los ítem contienen punteros los cuales señalan la posición en memoria en la cual se encuentra el valor de los dispositivos físicos y son los encargados de realizar cualquier tipo de petición del Cliente OPC, dígase obtener o modificar valores.

2.2.2 Tarjetas CRC

Las tarjetas CRC (Class, Responsibilities and Collaboration por sus siglas en inglés) sirven para diseñar el sistema en conjunto entre todo el equipo. Estas tarjetas como se aprecia en las Tablas 11 a la 15, representan objetos, para los cuales de especifican la clase a la que pertenece dicho objeto, las responsabilidades u objetivos que debe cumplir y las clases que colaboran con cada responsabilidad.

Responsabilidad	**Clases relacionadas**
Iniciar el cliente OPC	OPCClient

Detener el cliente OPC	OPCClient
Crear host.	OPCClient OPCHost

Tabla 11: Tarjeta CRC OPCClient.

Responsabilidad	**Clases relacionadas**
Obtener lista de servidores OPC locales.	OPCHost
Conectar al servidor Data Access	OPCHost OPCServer

Tabla 12: Tarjeta CRC OPCHost.

Responsabilidad	**Clases relacionadas**
Obtener Ítems del servidor.	OPCServer
Crear Grupo.	OPCServer OPCGroup
Obtener estado del servidor	OPCServer

Tabla 13: Tarjeta CRC OPCServer.

Responsabilidad	Clases relacionadas
Adicionar Ítem.	OPCGroup OPCItem
Habilitar/Deshabilitar la Lectura/Escritura Asincrónica.	OPCGroup
Cambiar el estado del grupo.	OPCGroup
Lectura sincrónica.	OPCGroup

Tabla 14: Tarjeta CRC OPCGroup.

Responsabilidad	Clases relacionadas
Insertar Ítem.	OPCItem
Habilitar/Deshabilitar Ítem.	OPCItem
Leer/Escribir Ítem	OPCItem
Eliminar Ítem	OPCItem

Tabla 15: Tarjeta CRC OPCÍtem.

2.2.3 Aplicación de patrones de diseño

Los patrones de diseño constituyen la base para la búsqueda de soluciones a problemas comunes en el desarrollo del software, ya que brindan una solución ya probada y

documentada a problemas de desarrollo de software que están sujetos a contextos similares. Para la descripción de los objetos y clases definidos, se utilizarán algunos patrones de diseños con el objetivo de solucionar un problema de diseño general.

Cuando se usan metodologías ágiles, como Extreme Programming (XP) y se centra el proceso en el desarrollo continuo, es inevitable elegir cuidadosamente las responsabilidades de cada clase en la primera codificación y fundamentalmente en la refactorización de dicho programa. Los patrones GRASP (General Responsability Assigment Software Patterns) constituyen patrones generales de software para asignación de responsabilidades, y son considerados como una serie de buenas prácticas de aplicación recomendable en el diseño de software. Entre ellos se destacan los que se mencionan a continuación.

Experto en información: Será utilizado debido a que es el principio básico de asignación de responsabilidades. El patrón indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento). Los objetos utilizan su propia información para llevar a cabo sus tareas Se distribuye el comportamiento entre las clases que contienen la información requerida. Son más fáciles de entender y mantener.

Un ejemplo del patrón experto se refleja en la clase Ítem, la cual es capaz de realizar todas las funcionalidades relacionadas con los ítem, en el siguiente fragmento de código (ver Figura #4), se puede observar como el método de la clase Ítem Escribir_Ítem, selecciona el ítem dado un nombre y el valor a modificar, busca el ítem y lo escribe síncronamente.

```
void OPC::Escribir_Item(std::string nomb_g, std::string nomb_item, QString text) {
    int pos = 0, pos2 = 0;
    for(int i=0; i<grupos.size(); i++) {
        if(grupos[i].get_NombreServidor().compare(nomb_g)) {
            pos = i;
            .
        }   .
    }
    VARIANT var;
    OPCItemData data;
    grupos[pos].get_Grupo().get_Items()[pos2]->writeSync(var);
            .
            .
}
```

Figura #4: Patrón experto, código fuente de la aplicación.

Controlador: Se asocia con operaciones del sistema y respuestas a sus eventos, tal como se relacionan los mensajes y los métodos. El controlador delega en otros objetos el trabajo que se necesita hacer pero coordina o controla la actividad. En otras palabras, el controlador no realiza estas actividades, sino que las delega en otras clases con las que mantiene un modelo de alta cohesión. Sirve como intermediario entre una clase interfaz y el algoritmo que la implementa ya que recibe los datos del usuario y la que los envía a las diversas clases según el método llamado.

El patrón controlador delega sus actividades en otras clases, puesto que dichas clases son las que contienen los elementos suficientes para realizar estas actividades. En el ejemplo reflejado en la Figura #5, se tiene a la clase controladora OPC que realiza una instancia de las demás clases y a través de las mismas delega sus responsabilidades.

```
void OPC::Desconectar_Servidor(std::string hostname, std::string nomb_serv) {
void OPC::Adicionar_Grupo(std::string hostname, std::string nomb_serv, std::string nomb_g) {
void OPC::Activar_Desactivar_Grupo(std::string nomb_g, bool activo) {
void OPC::Eliminar_Grupo(std::string nombre_g) {
void OPC::Adicionar_Item(std::string nomb_g, std::string nomb_item) {
void OPC::Activar_Desactivar_Item(std::string nomb_g, std::string nomb_item, bool activo) {
void OPC::Escribir_Item(std::string nomb_g, std::string nomb_item, QString text) {
void OPC::Eliminar_Item(std::string nomb_g, std::string nomb_item) {
```

Figura #5: Patrón controlador, código fuente de la aplicación.

Alta cohesión: Este patrón permite que cada elemento del diseño realice una única labor dentro del sistema sin que tenga mucha responsabilidad. Es decir, una clase tiene definida una responsabilidad en una determinada área que le permite colaborar con otras realizando así las tareas.

En el ejemplo de la Figura #6, se muestra como para la creación de un Ítem la clase server se apoya en la clase Grupo para la creación de dicho Ítem.

```
// Crear grupo
unsigned long refreshRate;
COPCGroup *grupo = opcServer->makeGroup("Grupo",true,1000,refreshRate,0.0);
CAtlArray<COPCItem *> items;

// Crear item
CString nomb_item = "Bucket Brigade.Int1";
COPCItem * item = grupo->addItem(nomb_item, true);
```

Figura #6: Patrón alta cohesión, código fuente de la aplicación.

Bajo acoplamiento: Asigna la responsabilidad de controlar el flujo de eventos del sistema a clases específicas, lo que facilita la centralización de actividades como validaciones y seguridad del sistema.

Si se analiza verticalmente la arquitectura que presenta el sistema, se puede observar que la clase Ítem en la capa del servidor solamente tiene dependencia de la clase grupo (ver Figura #7), y no tiene asociación de ningún tipo con la capa del cliente, logrando así la reusabilidad.

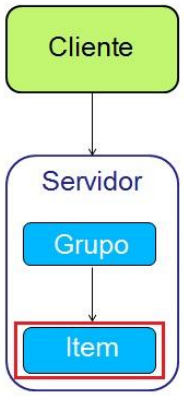

Figura #7: Patrón bajo acoplamiento, arquitectura del sistema.

Los patrones de diseño constituyen un conjunto de buenas prácticas, las cuales permiten facilitar el trabajo de los programadores, uno de los objetivos fundamentales del diseño.

2.3 Implementación

La implementación del sistema es la parte más importante dentro del desarrollo del proyecto en la metodología XP. Esta propone una serie de prácticas que sirven para el desarrollo exitoso del mismo, tales como el desarrollo de las iteraciones según el Plan de iteraciones, entrega en iteraciones pequeñas, un diseño simple y poco redundante del código con las funcionalidades necesarias estrictamente en el presente y las pruebas continuas, donde los programadores escriben las pruebas por cada unidad de código.

La implementación de forma iterativa de XP es una ventaja ya que al finalizar cada una se obtiene una versión del sistema superior a la anterior, donde se obtiene un producto funcional de la misma. En dicha funcionalidad se aplican pruebas al sistema las cuales pueden servir para retroalimentación del equipo de trabajo así como va encaminado el trabajo.

2.3.1 Tareas de programación por historia de usuarios

Las tareas de programación se definen con el objetivo de desglosar cada Historia de Usuario en tareas que serán desarrolladas por los programadores proporcionándoles una guía para un mejor desarrollo y cumplimiento de cada una de ellas.

Las tareas de programación son actividades que los programadores conocen que el sistema debe hacer. Deben ser estimables, y poder ser implementadas entre uno y tres

días ideales. La mayoría de las tareas de programación se derivan directamente de las historias de usuario (Ver Tabla 16).

Historias de Usuario	Tareas de ingeniería
Crear host.	Crear una interfaz para crear el host. Validar que el nombre del host sea el correcto.
Conectar al Servidor OPC.	Crear una interfaz para mostrar los distintos servidores existentes. Permitir conectar a un servidor. Permitir desconectarse del servidor.
Gestionar Grupo.	Crear una interfaz para crear el grupo. Validar que el nombre del grupo sea el correcto. Permitir habilitar un grupo. Permitir deshabilitar un grupo. Permitir eliminar un grupo.

Gestionar Ítem.	Crear una interfaz para crear el ítem. Validar que el nombre del ítem sea el correcto. Permitir habilitar un ítem. Permitir deshabilitar un ítem. Permitir eliminar un ítem. Permitir escribir un ítem. Permitir leer un ítem.
Conectar al servidor NVP.	Permitir enviar datos del cliente OPC hacia el servidor del NVP. Permitir escribir datos del servidor NVP hacia el cliente OPC.
Mostrar árbol de herencia.	Permitir mostrar interfaz con los distintos nodos conectados.

Tabla 16: Distribución de las tareas por cada Historia de Usuario.

2.3.2 Tareas de la Ingeniería

Las tareas de ingeniería son un conjunto de acciones a desarrollar para resolver las historias de usuario. Permiten organizar el proceso de implementación además de posibilitar que sea conocido el grado de complejidad de cada historia de usuario teniendo en cuenta la cantidad de tareas asociadas a ella (Ver Tablas de la 17 a la 36).

Tarea de ingeniería	
Número Tarea: 1.	**Número Historia de Usuario:** 1.
Nombre Tarea: Crear interfaz para la creación del host.	
Tipo Tarea: Desarrollo	**Puntos estimados (días):** 2.
Fecha inicio: 06/02/2013	**Fecha fin:** 06/02/2013

Programador responsable: Francisco Rubio Santiesteban.
Descripción: La aplicación muestra una interfaz en la cual solicita la entrada del nombre del host a crear.

Tabla 17: Tarea de ingeniería: Descripción Tarea #1.

	Tarea de ingeniería
Número Tarea: 2.	**Número Historia de Usuario:** 1.
Nombre Tarea: Validar que el nombre del host sea el correcto.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 2.
Fecha inicio: 08/01/2013.	**Fecha fin:** 10/01/2013.
Programador responsable: Francisco Rubio Santiesteban.	
Descripción: La aplicación verifica que el nombre del host sea correcto, puesto que solo puede ser el nombre de una pc en el dominio o su IP. En caso de estar incorrecto se notifica al usuario.	

Tabla 18: Tarea de ingeniería: Descripción Tarea #2.

	Tarea de ingeniería
Número Tarea: 3.	**Número Historia de Usuario:** 2.
Nombre Tarea: Crear una interfaz para mostrar los distintos servidores existentes.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 2.
Fecha inicio: 12/01/2013.	**Fecha fin:** 14/01/2013.
Programador responsable: Adrián Sánchez Parra.	
Descripción: La aplicación muestra una interfaz en la cual se listan los nombre de los servidores.	

Tabla 19: Tarea de ingeniería: Descripción Tarea #3.

	Tarea de ingeniería
Número Tarea: 4.	**Número Historia de Usuario:** 2.
Nombre Tarea: Permitir conectar a un servidor.	

Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 2.
Fecha inicio: 16/01/2013.	**Fecha fin:** 18/01/2013.
Programador responsable: Adrián Sánchez Parra.	
Descripción: La aplicación debe permitir al usuario seleccionar un servidor y conectarse al mismo.	

Tabla 20: Tarea de ingeniería: Descripción Tarea #4.

		Tarea de ingeniería
Número Tarea: 5.	**Número Historia de Usuario:** 2.	
Nombre Tarea: Permitir desconectarse del servidor.		
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 3.	
Fecha inicio: 19/01/2013.	**Fecha fin:** 22/01/2013.	
Programador responsable: Adrián Sánchez Parra.		
Descripción: La aplicación debe permitir al usuario desconectarse del servidor al cual se encuentra conectado.		

Tabla 21: Tarea de ingeniería: Descripción Tarea #5.

		Tarea de ingeniería
Número Tarea: 6.	**Número Historia de Usuario:** 3.	
Nombre Tarea: Crear interfaz para crear el grupo.		
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 3.	
Fecha inicio: 23/01/2013.	**Fecha fin:** 26/01/2013.	
Programador responsable: Francisco Rubio Santiesteban.		
Descripción: El usuario selecciona el servidor al cual se le agregará el grupo y la aplicación muestra una interfaz en la cual solicita la entrada del nombre del grupo a crear.		

Tabla 22: Tarea de ingeniería: Descripción Tarea #6.

		Tarea de ingeniería
Número Tarea: 7.	**Número Historia de Usuario:** 3.	

Nombre Tarea: Validar que el nombre del grupo sea el correcto.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 2.
Fecha inicio: 30/01/2013.	**Fecha fin:** 30/01/2013.
Programador responsable: Francisco Rubio Santiesteban.	
Descripción: La aplicación verifica que el nombre del grupo sea correcto, puesto que solo puede ser letras. En caso de estar incorrecto se notifica al usuario.	

Tabla 23: Tarea de ingeniería: Descripción Tarea #7.

	Tarea de ingeniería
Número Tarea: 8.	**Número Historia de Usuario:** 3.
Nombre Tarea: Permitir habilitar un grupo.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 3.
Fecha inicio: 31/01/2013.	**Fecha fin:** 04/02/2013.
Programador responsable: Francisco Rubio Santiesteban.	
Descripción: La aplicación debe permitir al usuario seleccionar un grupo deshabilitado y habilitarlo.	

Tabla 24: Tarea de ingeniería: Descripción Tarea #8.

	Tarea de ingeniería
Número Tarea: 9.	**Número Historia de Usuario:** 3.
Nombre Tarea: Permitir deshabilitar un grupo.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 3.
Fecha inicio: 05/02/2013.	**Fecha fin:** 08/02/2013.
Programador responsable: Francisco Rubio Santiesteban.	
Descripción: La aplicación debe permitir al usuario seleccionar un grupo habilitado y deshabilitarlo.	

Tabla 25: Tarea de ingeniería: Descripción Tarea #9.

Tarea de ingeniería

Número Tarea: 10.	**Número Historia de Usuario:** 3.
Nombre Tarea: Permitir eliminar un grupo.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 2.
Fecha inicio: 09/02/2013.	**Fecha fin:** 11/02/2013.
Programador responsable: Francisco Rubio Santiesteban.	
Descripción: La aplicación debe permitir al usuario seleccionar un grupo y eliminarlo.	

Tabla 26: Tarea de ingeniería: Descripción Tarea #10.

	Tarea de ingeniería
Número Tarea: 11.	**Número Historia de Usuario:** 4.
Nombre Tarea: Crear interfaz para crear el item.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 3.
Fecha inicio: 12/02/2013.	**Fecha fin:** 15/02/2013.
Programador responsable: Adrián Sánchez Parra.	
Descripción: El usuario selecciona el grupo al cual se le agregará el item y la aplicación muestra una interfaz en la cual solicita la entrada del nombre del item a crear.	

Tabla 27: Tarea de ingeniería: Descripción Tarea #11.

	Tarea de ingeniería
Número Tarea: 12.	**Número Historia de Usuario:** 4.
Nombre Tarea: Validar que el nombre del ítem sea el correcto.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 1.
Fecha inicio: 16/02/2013.	**Fecha fin:** 16/02/2013.
Programador responsable: Adrián Sánchez Parra.	
Descripción: La aplicación verifica que el nombre del ítem sea correcto. En caso de estar incorrecto se notifica al usuario.	

Tabla 28: Tarea de ingeniería: Descripción Tarea #12.

Tarea de ingeniería

Número Tarea: 13.	**Número Historia de Usuario:** 4.
Nombre Tarea: Permitir habilitar un ítem.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 2.
Fecha inicio: 18/02/2013.	**Fecha fin:** 20/02/2013.
Programador responsable: Adrián Sánchez Parra.	
Descripción: La aplicación debe permitir al usuario seleccionar un ítem deshabilitado y habilitarlo.	

Tabla 29: Tarea de ingeniería: Descripción Tarea #13.

	Tarea de ingeniería
Número Tarea: 14.	**Número Historia de Usuario:** 4.
Nombre Tarea: Permitir deshabilitar un ítem.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 2.
Fecha inicio: 21/02/2013.	**Fecha fin:** 23/02/2013.
Programador responsable: Adrián Sánchez Parra.	
Descripción: La aplicación debe permitir al usuario seleccionar un ítem habilitado y deshabilitarlo.	

Tabla 30: Tarea de ingeniería: Descripción Tarea #14.

	Tarea de ingeniería
Número Tarea: 15.	**Número Historia de Usuario:** 4.
Nombre Tarea: Permitir eliminar un ítem.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 2.
Fecha inicio: 26/02/2013.	**Fecha fin:** 28/02/ 2013.
Programador responsable: Adrián Sánchez Parra.	
Descripción: La aplicación debe permitir al usuario seleccionar un ítem y eliminarlo.	

Tabla 31: Tarea de ingeniería: Descripción Tarea #15.

	Tarea de ingeniería

Número Tarea: 16.	**Número Historia de Usuario:** 4.
Nombre Tarea: Permitir escribir un ítem.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 3.
Fecha inicio: 2/02/ 2013.	**Fecha fin:** 5/03/ 2013.
Programador responsable: Adrián Sánchez Parra.	
Descripción: La aplicación debe permitir modificar el valor del ítem seleccionado.	

Tabla 32: Tarea de ingeniería: Descripción Tarea #16.

	Tarea de ingeniería
Número Tarea: 17.	**Número Historia de Usuario:** 4.
Nombre Tarea: Permitir leer un ítem.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 3.
Fecha inicio: 6/03/ 2013.	**Fecha fin:** 9/03/ 2013.
Programador responsable: Adrián Sánchez Parra.	
Descripción: La aplicación debe permitir leer el valor del ítem seleccionado.	

Tabla 33: Tarea de ingeniería: Descripción Tarea #17.

	Tarea de ingeniería
Número Tarea: 18.	**Número Historia de Usuario:** 5.
Nombre Tarea: Permitir enviar datos del cliente OPC hacia el servidor NVP.	
Tipo Tarea: Desarrollo.	**Puntos estimados (días):** 2.
Fecha inicio: 13/03/2013.	**Fecha fin:** 16/03/2013.
Programador responsable: Francisco Rubio Santiesteban.	
Descripción: La aplicación debe permitir enviar los datos del cliente OPC hacia el servidor del NVP por medio de sockets.	

Tabla 34: Tarea de ingeniería: Descripción Tarea #18.

	Tarea de ingeniería
Número Tarea: 19	**Número Historia de Usuario:** 5.

Nombre Tarea: Permitir escribir datos del servidor NVP hacia el cliente OPC.	
Tipo Tarea: Desarrollo	**Tipo Tarea:** Desarrollo
Fecha inicio: 18/03/2013	**Fecha inicio:** 18/03/2013
Programador responsable: Francisco Rubio Santiesteban.	
Descripción: La aplicación debe permitir escribir datos enviados del servidor NVP hacia el cliente OPC por medio de sockets.	

Tabla 35: Tarea de ingeniería: Descripción Tarea #19.

	Tarea de ingeniería
Número Tarea: 20	**Número Historia de Usuario:** 6.
Nombre Tarea: Permitir mostrar interfaz con los distintos nodos conectados.	
Tipo Tarea: Desarrollo	**Puntos estimados (días):** 3.
Fecha inicio: 21/03/2013	**Fecha fin:** 24/03/2013
Programador responsable: Adrián Sánchez Parra.	
Descripción: La aplicación debe mostrar un árbol jerárquico en el que se muestran los diferentes nodos, dígase: host, servidores, grupos e ítems.	

Tabla 36: Tarea de ingeniería: Descripción Tarea #20.

2.3.3 Diagrama de Componentes

Los componentes son partes modulares del sistema, que pueden desplegarse y reemplazarse. Por lo general los componentes contienen clases y pueden ser implementados por uno o más artefactos. El diagrama de componentes entonces, muestra un conjunto de componentes relacionados entre sí. Su uso fundamental es estructurar el modelo de implementación y mostrar las relaciones de los elementos de implementación. El diagrama de componentes que se muestra en la Figura #9, representa las partes modulares en las que se divide el componente implementado.

En el diagrama se representan los principales componentes que se desarrollan, se estructuran en dos grupos a los cuales pertenecen Cliente o Servidor. Entre los componentes que se destacan se encuentran Servidores e Ítems.

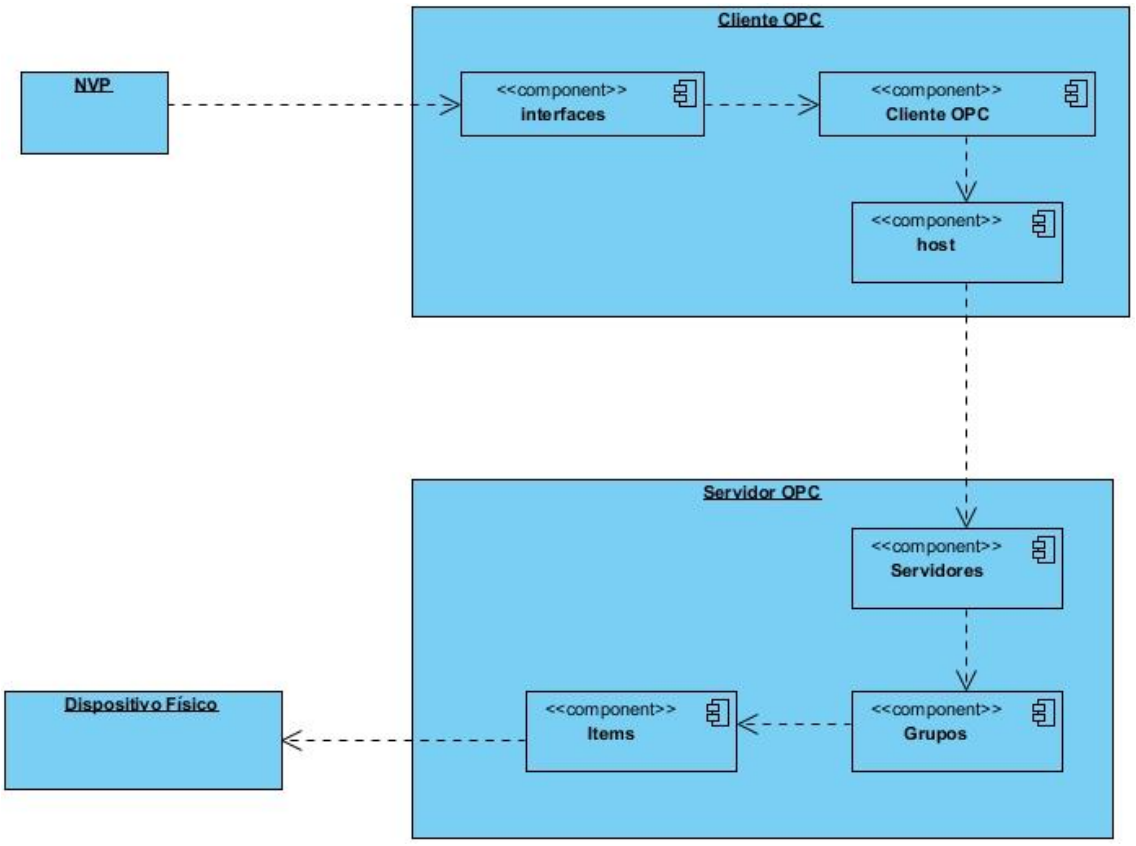

Figura #9: Diagrama de componentes.

2.3.4 Diagrama de Despliegue

El diagrama de despliegue muestra las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos, donde cada hardware se representa como un nodo. Las conexiones o relaciones de los nodos se etiquetan con un estereotipo que identifica el protocolo de comunicación utilizado. Este modelo es utilizado como base para la realización de las actividades de diseño e implementación.

En el nodo virtual como se muestra en la Figura #8, señalado de color verde los nodos que pertenecen al actual Nodo Virtual, y con color rojo los nodos que se integran a la aplicación. En la integración se adicionan tres nodos. El primero es el Cliente OPC que conecta con el servidor del Nodo Virtual por medio de socket y un puerto. Con el Cliente OPC se conecta el Servidor OPC, y estos se conectan a través de un protocolo y un puerto, el cual está definido por las aplicaciones. El servidor OPC está conectado con los dispositivos físicos por medio de USB.

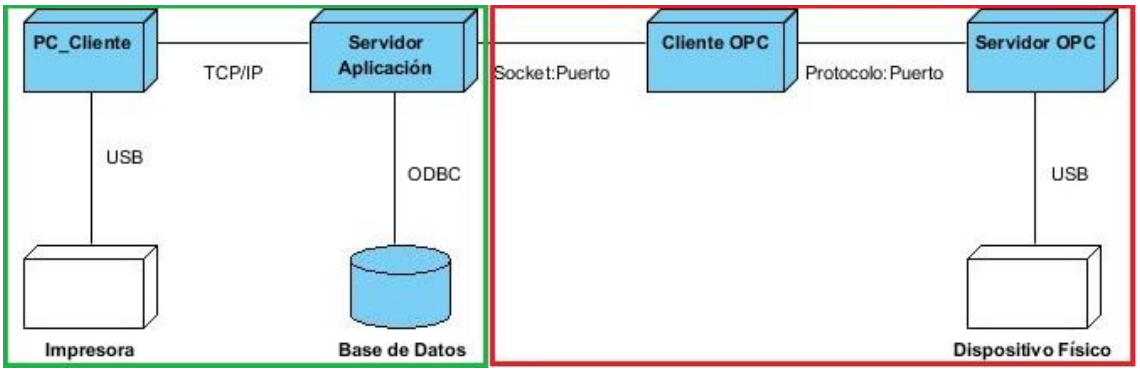

Figura #8: Diagrama de Despliegue.

2.4 Conclusiones Parciales

En el presente capítulo se confeccionaron los artefactos del análisis y el diseño que le dan solución a la problemática planteada. Se desarrollaron las historias de usuarios para la especificación de los requisitos funcionales del sistema, de las cuales se desarrollaron un total de 16 funcionalidades, reflejadas en nueves historias de usuarios para lograr la inserción de los dispositivos físicos en la aplicación. Además se desarrolló el plan de iteraciones en el cual serán implementadas en cada iteración las historias de usuarios con mayor grado complejidad para el negocio.

Se diseñaron un total de cinco tablas CRC, en las cuales quedó plasmado el diseño de la aplicación. Se aplicaron un total de seis patrones de diseño, demostrando la flexibilidad de las clases y la robustez del sistema. También se desarrollaron un total de 20 tareas de la ingeniería para la especificación de las acciones a realizar a la hora de programar. También quedaron plasmados los diagramas de componentes y despliegue con el objetivo de proveer a los clientes de una mejor visión de los elementos por los que se encuentra formada la aplicación.

CAPÍTULO 3: VALIDACIÓN

Introducción

El proceso de pruebas constituye uno de los pilares de la metodología XP. Esta metodología propone que se realicen pruebas constantemente, pues del buen uso de dichas pruebas depende el éxito. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos no deseados a la hora de realizar modificaciones y refactorizaciones.

3.1 Métricas para la validación de requisitos

Para medir la calidad de la especificación de requisitos de software, se proponen cuatro métricas: especificidad (ausencia de ambigüedad), compleción, corrección, comprensión, y la capacidad de verificación. (Overmyer, 1993)

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 (10) y no funcionales (13) se obtuvieron 23 requisitos en total.

3.1.1 Especificidad

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)

$$Q1 = \frac{Rii}{Rt} = \frac{23}{23} = 1$$

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

Rt : Total de los requisitos.

3.1.2 Compleción

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

$$Q2 = \frac{na}{na + nb} = \frac{23}{23 + 0} = 1$$

na : Número de requisitos funcionales completos.

nb : Número de requisitos funcionales pobremente especificados.

3.1.3 Corrección

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:

$$Q3 = \frac{Rc}{Rc + Rnv} = \frac{23}{23 + 0} = 1$$

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

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

3.1.4 Comprensión

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

$$Q4 = \frac{Rbc}{Rt} = \frac{23}{23} = 1$$

Rbc : Número de requisitos que todos los revisores entienden.

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.

3.2 Métricas para la validación del diseño

Las métricas de software constituyen los elementos que permiten evaluar la calidad de una determinada característica o artefacto que se genere en un proyecto de software. En este epígrafe se validará la solución propuesta mediante la aplicación de métricas que permitirán medir el estado de algunos atributos de calidad que facilitarán expresar una opinión valorativa sobre el diseño y la calidad de los componentes obtenidos.

Con la aplicación de estas métricas se abarcan los siguientes atributos de calidad:

- **Cantidad de pruebas:** consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad del producto (clase, conjunto de clases, componente, módulo, entre otros.) diseñado.
- **Responsabilidad o Cohesión:** consiste en la responsabilidad asignada a una clase modelada de un dominio o concepto, de la problemática propuesta.
- **Complejidad de implementación:** consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- **Complejidad del mantenimiento:** consiste en el grado de esfuerzo necesario a

realizar para desarrollar una reparación, una mejora o una corrección de algún error de un diseño de software. Influye fuertemente en los costes y la planificación del proyecto.

- **Reutilización:** consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento:** consiste en el grado de dependencia o interconexión de una clase o estructura de clase con otras, está muy atada a la característica de Reutilización.

Para comprobar los atributos antes mencionados se seleccionaron dos técnicas de medición o métricas, Tamaño Operacional de las Clases y Relaciones entre Clases, las cuales se abordan en los siguientes dos subepígrafes.

3.1.1 Tamaño Operacional de las Clases (TOC)

El tamaño general de una clase se ajusta a la realización de un cálculo de sus atributos u operaciones totales.

- El total de operaciones (operaciones tanto heredadas como privadas de la instancia), que se encapsulan dentro de la clase.
- El número de atributos (atributos tanto heredados como privados de la instancia), encapsulados por la clase.

Los valores grandes para la métrica TOC indican que la clase debe tener bastante responsabilidad, lo que propicia un bajo nivel de reutilización de la clase y complicará la implementación y las pruebas. En general, operaciones y atributos heredados o públicos deben ser ponderados con mayor importancia cuando se determina el tamaño de clase, pues las operaciones y atributos privados, permiten la especialización.

También se pueden calcular los promedios para el número de atributos y operaciones de cada clase. Cuanto más pequeño sea el valor promedio para el tamaño, será más viable para que las clases dentro del sistema puedan reutilizarse. Durante la realización de este trabajo se decidió aplicar la métrica para el tamaño de clases en función de sus operaciones (TOC). Luego de aplicar dicha métrica a las clases de la solución la misma arrojó los siguientes resultados (ver Figuras 10 a la 12):

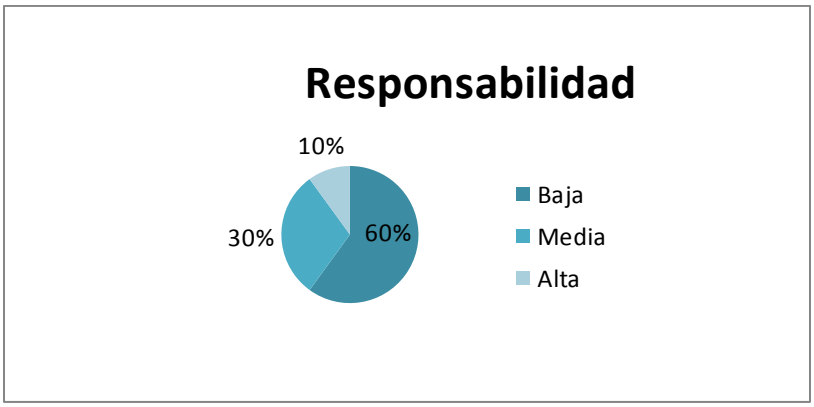

Figura #10: Representación de la incidencia de los resultados dela evaluación de la métrica TOC en el atributo Responsabilidad.

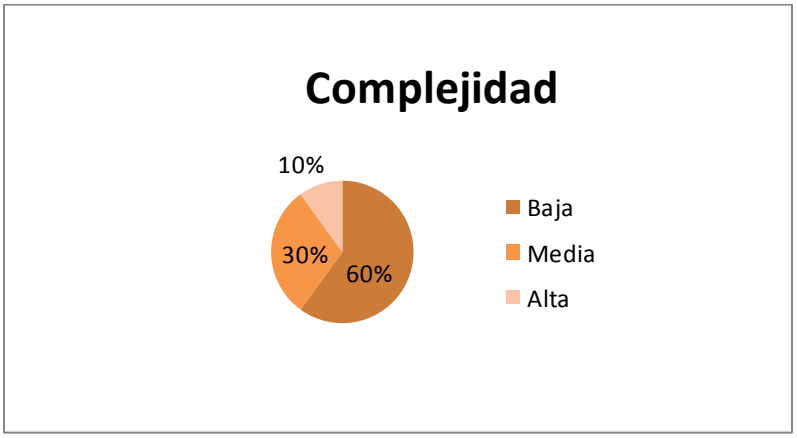

Figura #11: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de implementación.

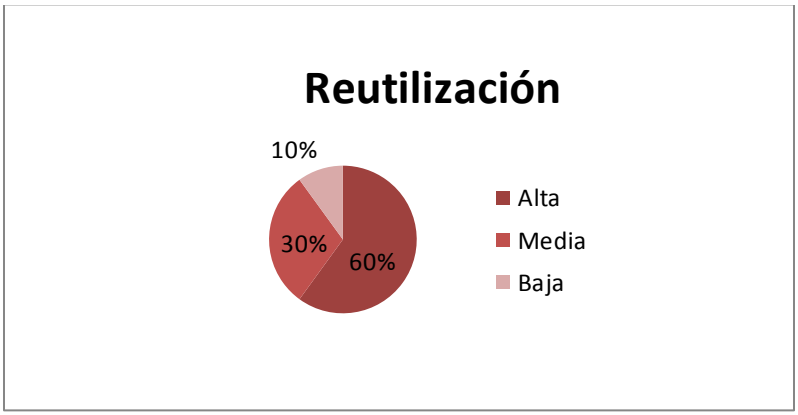

Figura #12: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Después de aplicada la métrica se puede concluir que el diseño realizado en el sistema tiene una buena calidad, pues se obtuvo resultados satisfactorios en los diferentes atributos de calidad medidos, por lo que se puede afirmar que se obtiene una solución eficiente con altos niveles de reutilización (60%) lo que garantiza que los componentes

elaborados puedan servir de ayuda a otro proyectos de software. También se evidencia que el diseño realizado fue correcto ya que se obtuvieron bajos niveles de responsabilidad (60%) y complejidad de implementación (60%), pues se le asignaron correctamente las responsabilidades a las clases involucradas en la solución.

3.1.2 Relaciones entre clases

El resultado de la aplicación de la métrica RC está dado por el número de relaciones de uso de una clase con otras. La aplicación de dicha métrica permite evaluar atributos como el Acoplamiento, la Complejidad de mantenimiento, la Reutilización y la Cantidad de pruebas. De forma tal que mientras mayor sea las relaciones de uso entre clases mayor será el Acoplamiento, la Complejidad de Mantenimiento y la Cantidad de pruebas, mientras que su Reutilización disminuye. Después de aplicar dicha métrica se obtuvieron los siguientes resultados (ver Figuras 13 a la 16):

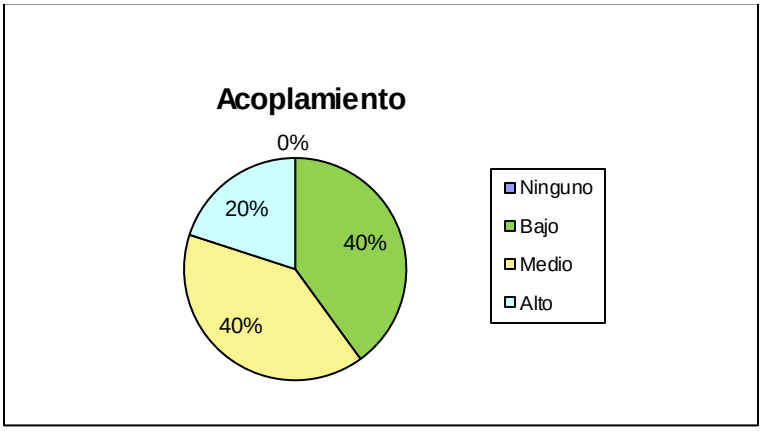

Figura #13: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.

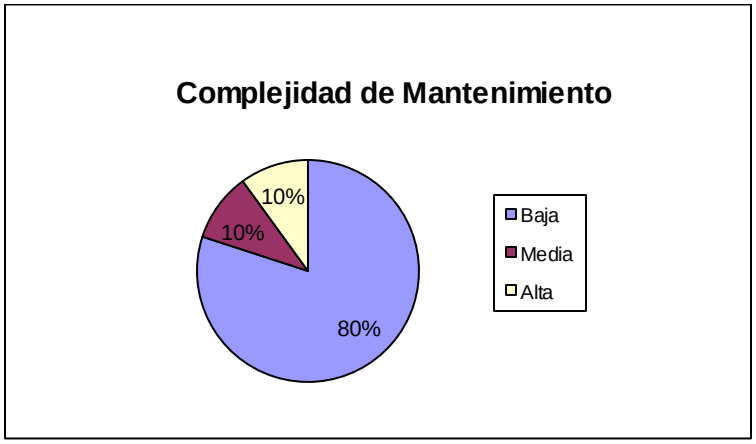

Figura #14: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo

Complejidad de Mantenimiento.

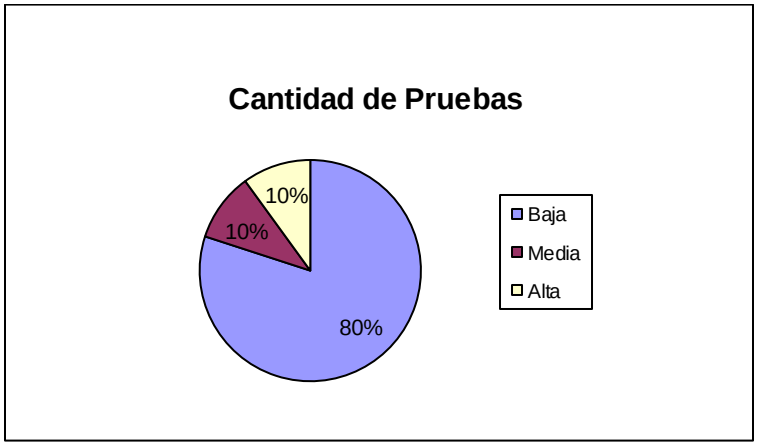

Figura #15: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

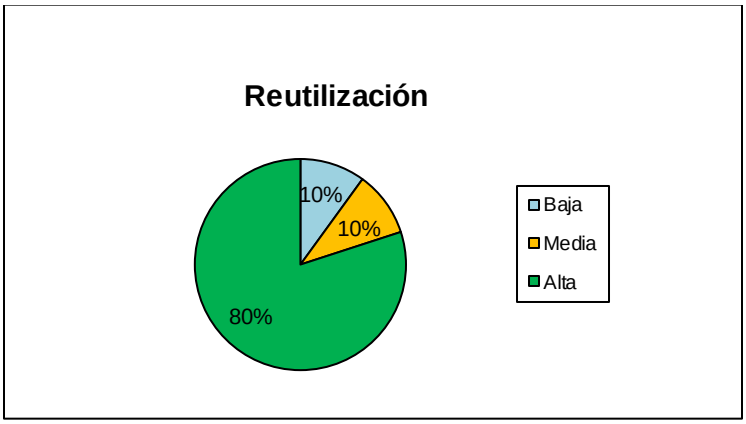

Figura #16: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización.

La aplicación de esta métrica de software demostró que existe un bajo acoplamiento entre las diferentes clases de la solución propuesta con un 40% y con un 80% de complejidad de mantenimiento y reutilización. Se demostró que dichas clases poseen un bajo acoplamiento lo que posibilita una fácil desarticulación da las partes de los componentes desarrollados a la hora de reutilizar los mismos. También arrojó resultados satisfactorios en el atributo Complejidad de mantenimiento lo que facilita las tareas de corrección, modificación y mantenimiento de los componentes.

3.3 Pruebas unitarias.

Como resultado del capítulo uno se muestra el resultado de las pruebas aplicadas al

código de la aplicación o pruebas unitarias.

3.3.1 Camino Básico

En la validación de la solución de la aplicación, se realizó la prueba del camino básico a las funcionalidades de mayor complejidad. Esta permite definir los diferentes caminos independientes de la función y probar su funcionamiento al menos una vez. En el siguiente código de ejemplo representado en la Figura #16, se muestra el proceso desarrollado al método “Activar_Desactivar_Grupo” de la Clase controladora “OPC.cpp”.

Para ello se comienza por analizar el código y enumerar las instrucciones:

```
void OPC::Activar_Desactivar_Grupo(std::string nomb_g, bool activo) {
    int pos = 0;                                                    //1
    for(int i=0; i<grupos.size(); i++) {                            //2
        if(grupos[i].get_NombreGrupo().compare(nomb_g)) {           //3
            pos = i;                                                //4
            break;                                                  //4
        }                                                           //4
    }                                                               //5
    grupos[pos].get_Grupo().set_Activo(activo);                     //6
}                                                                   //6
```

Figura #16: Función a la que se le aplica el método del Camino Básico.

Seguidamente es necesario representar el grafo de flujo (o grafo del programa), que va a representar el flujo de control lógico del código anterior, a través de nodos, aristas y regiones. Quedando como muestra la Figura # 17.

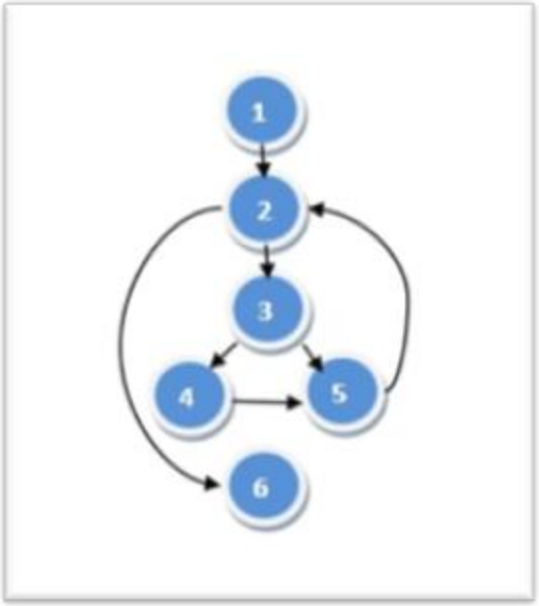

Figura #17: Grafo de flujo, dada la función de la figura #16.

Luego de realizado el grafo es necesario conocer la cantidad de caminos independientes que se deben buscar para probar; y para esto se calcula la complejidad ciclomática, que se puede calcular de tres formas:

1. El número de regiones del grafo de flujo coincide con la complejidad ciclomática. Por lo que V (G) = 3.

2. V (G) = (A –N) + 2 donde “A” es el número de aristas del grafo de flujo y “N” es el número de nodos del mismo. V (G) =7 –6 +2 = 3.

3. V (G) =P+ 1 donde “P” es el número de nodos predicado contenidos en el grafo.

Los nodos 2 y 3 son nodos predicados. V (G) = 2+1 = 3.

Como se muestra en cualquiera de las tres formas anteriores la complejidad ciclomática es 3, por lo que la cantidad de caminos básicos que puede tomar el algoritmo durante su ejecución es 3 y quedan definidos así:

Camino básico #1: 1 –2–6.

Camino básico#2: 1 –2 –3 –5 –2 –6.

Camino básico #3: 1 –2 –3 –4 –5 –2 –6.

Ya seguidamente se preparan los casos de pruebas que forzarán la ejecución de cada camino del conjunto básico, quedando de la siguiente forma:

Descripción: Se deben activar o desactivar el grupo, dado el nombre del grupo (nomb_g) y el estado (activo) en que se encuentra “Activar_Desactivar_Grupo”. Esta descripción es común para todos los casos de prueba que se realizarán.

Caso de prueba para el camino básico # 1.

Condición de ejecución: Para esta prueba es necesario que la lista “grupos” no reciba datos, se encuentre vacía.

Entrada: La lista “grupos” vacía.

Resultados esperados: Teniendo en cuenta la condición de ejecución se espera que el estado del grupo resulte el mismo.

El resultado obtenido fue correcto.

Caso de prueba para el camino básico # 2.

Condición de ejecución: Para esta prueba es necesario que la lista "grupos" contenga elementos, pero que no esté incluido el grupo que se desea modificar el estado.

Entrada: La lista "grupos" contiene una cantidad de elementos, en la cual no está incluido el grupo.

Resultados esperados: Teniendo en cuenta la condición de ejecución se espera que chequee todos los grupos de la lista, sin embargo el estado del grupo no se modifica.

El resultado obtenido fue correcto.

Caso de prueba para el camino básico # 3.

Condición de ejecución: Para esta prueba es necesario que la lista "grupos" contenga elementos y además esté incluido el grupo que se desea modificar el estado.

Entrada: La lista "grupos" contiene una cantidad de elementos en la cual está incluido el grupo.

Resultados esperados: Teniendo en cuenta la condición de ejecución se espera que chequee todos los grupos de la lista y cuando encuentre el nombre del grupo que se corresponda con el del parámetro, se modifique.

El resultado obtenido fue correcto.

3.4 Pruebas funcionales

Como cumplimiento al capítulo 1, se realizaron las pruebas de caja negra, por medio de la técnica de partición de equivalencia.

3.4.1 Técnica de partición de equivalencia

La técnica de partición de equivalencia *"es un método de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba."* Pressman (Quinta edición). Los casos de pruebas que se diseñan para este tipo de técnicas se basan en una evaluación de las clases de equivalencia para una condición de entrada. *"Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada."* Pressman (Quinta edición).

Para la definición de clases de equivalencia Pressman en su quinta edición señaló las siguientes directrices:

- *Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos no válidas.*

- *Si una condición de entrada requiere un valor especifico, se define una clase de equivalencia válida y una no válida.*
- *Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una no válida.*
- *Si una condición de entrada es lógica, se define una clase de equivalencia válida y una no válida.*

Para crear los casos de prueba a partir de las clases de equivalencia se han de seguir los siguientes pasos:

- Asignar a cada clase de equivalencia un número único.
- Hasta que todas las clases de equivalencia hayan sido cubiertas por los casos de prueba, se tratará de escribir un caso que cubra tantas clases válidas no incorporadas como sea posible.

Hasta que todas las clases de equivalencia no válidas hayan sido cubiertas por casos de prueba, escribir un caso para cubrir una única clase no válida no cubierta. (Huerta, 2009)

En la detección de las no conformidades se realizaron tres iteraciones, en las que se detectaron un total de 29 no conformidades, detectándose 20 en una primera iteración, lo cual equivale a un 70 por ciento, después de darle solución se prosiguió a una segunda iteración. En una segunda iteración se identificaron un total de 7 no conformidades, para un 25 por ciento a las cuales se le dio solución y por último se realizó una tercera iteración con un total de 2 no conformidades detectadas para un 5 por ciento, las cuales también se le dieron solución, para una mayor comprensión ver figura #18 a continuación.

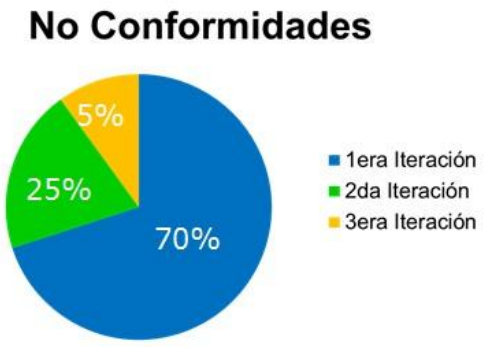

Figura #18: Gráfico de no conformidades.

Dentro de las principales no conformidades que se detectaron se pueden señalar, errores ortográficos, no se muestran mensajes de alerta en caso de errores o campos vacíos, no se verifica la duplicidad de los elementos y no se agregan componentes después de

crearlos. Como resultado después de la tercera iteración quedaron resueltas todas las no conformidades detectadas.

3.5 Conclusiones Parciales

En el presente capítulo se obtuvieron los resultados de las pruebas realizadas a la herramienta, tanto en la fase de análisis, diseño e implementación, obteniéndose resultados satisfactorios.

Se aplicaron un total de cuatro métricas en la validación de los requisitos, las cuales evidenciaron que los requisitos no tienen ambigüedad, además poseen buena comprensión, corrección y compleción.

En cuanto a la validación del diseño se aplicaron dos métricas, las métricas TOC y RC las cuales arrojaron valores de baja responsabilidad, complejidad, acoplamiento y cantidad de pruebas y valores de alta reutilización.

Para la validación del código de la aplicación se realizó el método del camino básico, logrando definir los diferentes caminos independientes de las funciones y probar su funcionamiento al menos una vez. También se realizaron pruebas a las interfaces por medio de la técnica de partición de equivalencia, arrojando como resultado un total de 29 no conformidades, las cuales fueron resueltas.

La aplicación de dichas métricas y técnicas proporcionó modelos efectivos de análisis y diseño, así como la implementación de un código sólido y pruebas exhaustivas.

CONCLUSIONES GENERALES

Se seleccionó como estándar de comunicación industrial, el OPC para la incorporación de los dispositivos físicos en el Nodo Virtual.

Se desarrollaron un total de 16 funcionalidades, reflejadas en nueve historias de usuarios para lograr la inserción de los dispositivos físicos en la aplicación.

Se diseñaron un total de cinco tablas CRC, en las cuales quedó plasmado el diseño de la aplicación.

Se aplicaron un total de seis patrones de diseño, demostrando la flexibilidad de las clases y la robustez del sistema.

Se aplicaron un total de seis métricas y dos técnica para la validación de la aplicación, las cuales arrojaron resultados satisfactorios.

RECOMENDACIONES

Desarrollo de otros protocolos de comunicación, en la herramienta Nodo Virtual de Procesos.

BIBLIOGRAFÍA

[1]. Gámez Batista, Ing. Yalice. 2008. Herramienta interactiva para la enseñanza en la carrera de Ingeniería Automática: Nodo Virtual de Procesos. 2008.

[2]. Castelán, Dr. Mario. 2006. Métodos cuantitativos y simulación. Introducción al modelado y simulación de sistemas. 2006.

[3]. Shannon, Robert E. Introduction to Simulation. 1992.

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

[5]. Hugh J. Watson John H. Blackstone Jr. Wiley & Sons, Inc. Singapore. Computer simulation. 1989

[6]. Penner, D. "Cognition, computers, and synthetic science: building knowledge and meaning trough modeling". (2001)

[7]. Ortiz, Leydis A y Edith, Maylen. Análisis y Diseño de un Nodo Virtual de Procesos. Universidad de las Ciencias Informáticas. Habana: s.n., 2008. pág. 169.

[8]. Barberán, Manuel. Funcionamiento del Protocolo TCP-IP. 1998.

[9]. Tolfo, Flavio. Cubriendo un hueco en el mercado de buses de campo. ABB, Vol. 1, págs. 30-34. 2002

[10]. Petzold, Charles. Programación en Windows 95. Madrid, España: McGraw-Hill, 1996. pág. 399.

[11]. Inc, Matrikon. MatrikonOPC.[En línea] Matrikon Inc, 2010. [Citado el: 5 de Mayo

[12]. de 2010.] www.matrikonopc.com/

[13]. Jacobson, Ivar, Booch, Grady y Rumbaugh, James. El Proceso Unificado de desarrollo de Software. Madrid: Addison Wesley, 1999.

[14]. Microsoft. Visual C++. MSDN. [En línea] Microsoft, 2010. [Citado el: 1 de Mayo de 2010.] www.msdn.microsoft.com/es-es/library/60k1461a(v=VS.80).aspx.

[15]. Noble, Angela Martin y Robert Biddle, James. The XP Customer Role in Practice: Three Studies Agile. 2004.

[16]. Beck, Kent. "Extreme Programming Explained. Embrace Change", Pearson Education, 1999. Traducido al español como: "Una explicación de la programación extrema. Aceptar el cambio", Addison Wesley, 2000.

[17]. Jeffries, R., Anderson, A., Hendricks on, C. "Extreme Programming Installed".

[18]. Addison-Wesley. 2001

[19]. Newkirk, J., Martin R.C. "Extreme Programming in Practice". Addison-Wesley.2001.

[20]. Wake, W.C “Extreme Programming Explored”. Addison-Wesley. 2002.

[21]. M, S Pérez. Propuesta de modelo de desarrollo de software tecnológico del Centro de Soluciones de Gestión. Cuidad de la Habana: s.n., 2009.

[22]. Jacobson, Ivar, Booch, Grady y Rumbaugh, James. “El Proceso Unificado de Desarrollo de Software”. 2000: s.n.

[23]. Don Wells. 2009. Extreme Programming. [En línea] 2009 www.extremeprogramming.org/.

[24]. RAE, www.buscon.rae.es/drae/srv/search?id=lPyNMvlaDDXX22cDL6nd

[25]. RAE, 2001. Edición 22. www.buscon.rae.es/drae/srv/search?id=lPyNMvlaDDXX22cDL6nd

[26]. MODBUS, 2001 www.modbus.org

[27]. PROFIBUS, 2011 www.smar.com/espanol/profibus.asp

[28]. OPCCONNECT, 2011 www.opcconnect.com/history.php

[29]. MatrikonOPC, 2013 http://www.matrikonopc.com/resources/case-studies.aspx

[30]. Huerta Carralero, I. Propuesta para la realización de pruebas en un proyecto de software [Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas]. [La Habana]: Universidad de las Ciencias Informáticas; 2009.

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