UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
Facultad 6

TÍTULO: Solución informática para la comunicación de QGIS con aplicaciones externas.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICA

Autor: Dayana Silveira Romero.

Tutor: MSc David Silva Barrera.

Ciudad de La Habana, junio, 27 del 2011
"Año 53 de la Revolución"

Camina hacia el futuro, abriendo nuevas puertas y probando cosas nuevas, se curioso... porque nuestra curiosidad siempre nos conduce por nuevos caminos.

Walt Disney

Dedicatoria

A mis padres y mis tres preciosos hermanos por ser mi fuerza y mi confianza.

AGRADECIMIENTOS

Agradecimientos

Agradezco a mi mamita por todo su esfuerzo, por quererme tanto, y ser tan especial e incondicional conmigo. Gracias por tu confianza, por ser amiga, madre y guía, por el apoyo recibido durante mi formación profesional.

A mi papito, por su amor y por saber exigirme en mis estudios. Gracias a su apoyo y consejos he llegado a realizar la más grande de mis metas. La cual constituye la herencia más valiosa que pudiera recibir.

A Oscarito mi segundo papá, por ser mi ídolo como profesional culto y dedicado al trabajo.

A mis padres en general no existirá una forma de agradecer una vida de sacrificio y esfuerzo, quiero que sientan que el objetivo logrado también es de ustedes y que la fuerza que me ayudó a conseguirlo fue su apoyo.

A mis hermanitos bellos, Dailyn, Albertico y Danielita por darme tanto amor y aunque son tan pequeños hacen cosas grande…son mi luz, lo más hermoso que tengo en la vida, mi razón de ser. Sus presencias han sido y

será siempre el motivo más grande que me ha impulsado para lograr esta meta.

A mi novio Osmany por estar siempre a mi lado ayudándome a conquistar este fin. Por ser tan comprensivo y dar lo mejor de sí sin esperar nada a cambio… por saber escuchar y brindarme su apoyo cuando es necesario, por todo gracias.

A mis abuelos Minaelia, Matilde y papá Niñito, que siempre se han preocupado tanto por mí, gracias por su apoyo a través de mis estudios y con la promesa de seguir siempre adelante.

A mi prima Leidiana por ser siempre mi amiga y mi confidente, por escucharme y aconsejarme cuando lo necesito.

A Yary por darme una hermana tan hermosa y cuidar tanto de mi papito.

A Danelita, tía Aleida, tío Pipo, Robertico, Isnel y Cuco, a mi tía Marta, Misleidis, Gallega, Gustavito y Robertico. A mi familia en general muchas gracias por su confianza.

AGRADECIMIENTOS

A mis vecinos y amistades de Güira por estar siempre preocupados por mis resultados.

A mis eternos amigos Rolando y Gretel.

Agradezco a mis compañeros de estudios del IPVCE Lianet, Soles, Marbelis, Osmanis, Evelio, en general a todos que aunque pasamos momentos difíciles juntos logramos vencer todas las metas, gracias por su lealtad y cariño.

A mis amigos de primer año Frank y Lisandra. A mis amigas de siempre Rocio, Susana, Harlenia y Yaima.

A mis nuevas amigas Yasnary y Danieyis por ser tan especiales conmigo y estar siempre a mi lado, gracias por su preocupación y estar siempre que las necesito.

Gracias a mis compañeros de proyecto y profesores por su ayuda. A mi tutor por el tiempo dedicado.

A TODOS GRACIAS.

DECLARACIÓN DE AUTORÍA

Declaración de Autoría

Ciudad de La Habana, junio, 2011
"Año 53 de la Revolución"

Declaro que soy la única autora de este trabajo y autorizo a la Universidad de las Ciencias Informáticas (UCI) a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del 2010.

Dayana Silveira Romero.

_____
FIRMA DEL AUTOR

Msc. David Silva Barrera.

_____
FIRMA DEL TUTOR

Datos de Contacto

Autor

Nombre: Dayana.

Apellidos: Silveira Romero.

Correo electrónico: dsilveira@estudiantes.uci.cu

Tutor

Nombre: David.

Apellidos: Silva Barrera.

Correo electrónico: dsilva@uci.cu

Categoría Docente: Máster en Ciencias.

Resumen

En centro de desarrollo de GeoInformática y Señales Digitales de la Universidad de Ciencias Informáticas se lleva a cabo un proyecto para la implementación de un Sistema de Información Geográfica en plataforma de escritorio a través de la herramienta **Quantum GIS**, dirigido a personalizar y satisfacer las necesidades de representación y análisis cartográfico. En este SIG de escritorio se hace necesario un módulo que automatice la integración con aplicaciones externas, que permita la representación y el procesamiento de información provenientes de bases de datos cartográficas.

Para ello se hace un estudio teórico de aspectos relacionados con los Sistemas de Información Geográfica, donde quedan reflejados los conceptos fundamentales para entender el funcionamiento del entorno de trabajo y el flujo de la información en los mismos. Además se realiza un análisis de las tendencias y tecnologías existentes, valorando los principales sistemas de comunicación más conocidos y avalados internacionalmente, proponiéndose los que se consideran más adecuados para el desarrollo.

Se realiza todo un proceso de ingeniería que comenzando por el levantamiento de requisitos y pasando por la realización de diagramas de casos de usos, especificación de casos de usos, diagramas de clases y diagramas de arquitecturas permiten presentar un modelo adecuado de la aplicación a desarrollar. Se implementa el sistema acreditando los resultados sobre la base de pruebas realizadas al software, a partir de los casos de pruebas definidos, validándose un adecuado estándar de codificación y una correcta gestión de configuración durante todo el ciclo de vida del software.

PALABRAS CLAVES

SIG, Quantum GIS, Comunicación, Middleware, ICE, Plugin, Modelo, Ingeniería.

ÍNDICE

Tabla de Contenidos

Dedicatoria III

Agradecimientos IV

Declaración de Autoría VII

Datos de Contacto VIII

Resumen IX

Introducción 1

Capítulo 1: Fundamentación Teórica 4

1.1. Conceptos asociados al dominio del problema 4
1.2. Objeto de estudio 6
1.3. Análisis de tecnologías que permiten la comunicación entre aplicaciones. ... 9
1.3.1. Solución basada en RPC (*Remote Procedure Call* en inglés/ llamada a un procedimiento remoto) 9
1.3.2. Solución basada en The Internet Communications Engine (ICE) 11
1.3.3. Solución basada en Socket 12
1.3.4. SOAP (Simple Object Access Protocol) 13
1.4. Conclusiones 15

Capítulo 2: Tendencias y tecnologías actuales a utilizar. 16

2.1. Arquitectura de Comunicación 16
Comunicación Punto a Punto (P2P) 16
2.2. Tecnología de comunicación 17
Internet Communications Engine (ICE) 17
2.3. Lenguajes de Programación (C++) 18
2.4. El Lenguaje Unificado de Modelado (UML) como soporte de la modelación de la solución propuesta 19

2.5. Metodología de desarrollo del Software 19
El Proceso Unificado de Desarrollo de Software (RUP).....20
2.8. Herramienta CASE para el modelado 21
Visual Paradigm 21
2.10. Entornos de Desarrollo Integrado (IDE) 22
QT Creator 22
2.11. Conclusiones. 22

Capítulo 3: Presentación de la solución propuesta.....23

3.1. Solución propuesta 23
3.2. Modelo de Dominio 23
3.1.1. Representación del Modelo de Dominio 24
3.1.2. Definición de las clases del Modelo de Dominio 24
3.3. Requisitos funcionales 25
3.4. Requisitos no funcionales 26
3.5. Modelo de Casos de Usos del Sistema.....27
3.5.1. Descripción de los Actores 28
3.5.2. Diagrama de Casos de Usos del Sistema.....29
3.5.3. Descripción textual de los Casos de Usos del Sistema.....29
3.6. Conclusiones 34

Capítulo 4: Construcción de la Solución Propuesta.....35

4.1. Principios de Diseño 35
4.2. Arquitectura Lógica del Middleware 35
4.3. Diagrama de Clases del Diseño.....36
4.4. Patrones de Diseño.....38
4.5. Seguridad del Middleware.....40
4.6. Generalidades de la Implementación 41
4.7. Estándar de codificación 43
4.8. Prueba del sistema propuesto.....45
4.9. Conclusiones 51

ÍNDICE

Conclusiones Generales 52

Recomendaciones 53

Bibliografía Referenciada 54

Bibliografía Consultada 56

Glosario de Términos 57

Anexo I: Tecnologías de Comunicación entre procesos 59

ÍNDICE DE TABLAS

Tabla 1 Descripción de los actores del sistema 28
Tabla 2 Descripción del CU Configurar comunicación 30
Tabla 3 Descripción del CU Solicitar servicio Cargar Cartografía 31
Tabla 4 Descripción del CU Realizar Solicitud Zoom 33
Tabla 5 Descripción del CU Ordenar Cambio Cartografía 34
Tabla 6: Sección a probar en el Caso de Uso Establecer Comunicación. 47
Tabla 7: Matriz de datos Comunicar QGIS 47
Tabla 8: Sección a probar en el Caso de Uso Solicitar servicio Cargar Cartografía 48
Tabla 9: Matriz de datos Cargar Cartografía 48
Tabla 10: Sección a probar en el Caso de Uso Realizar solicitud Zoom 49
Tabla 11: Descripción de las variables para el caso de uso Realizar solicitud Zoom 50
Tabla 12: Matriz de datos Realizar Zoom. 50
Tabla 13: Sección a probar en el Caso de Uso Ordenar Cambio Cartografía. 51
Tabla 14: Matriz de datos Ordenar cambio de Cartografía. 51

ÍNDICE DE FIGURAS

Figura 1: Ejemplo de Plugins.....9
Figura 2: Modelo de dominio.....24
Figura 3 Diagrama de Casos de Usos del Sistema.....29
Figura 4 Diagrama de Paquetes del Middleware.....36
Figura 5 Diagrama de Clases del Diseño Establecer Comunicación.....36
Figura 6 Diagrama de Clases del Diseño del CU Solicitar Servicio Cargar Cartografía 37
Figura 7 Diagrama de Clases del Diseño del CU Realizar Solicitud Zoom.....37
Figura 8 Diagrama de Clases del Diseño del CU Ordenar Cambio Cartografía.....38
Figura 11 Diagrama de despliegue.....42
Figura 12 Diagrama de Componentes.....43

Introducción

Los últimos diez años han sido testigo de avances masivos en las áreas de la computación, la primera es que el hardware se ha ido abaratando cada vez más, y a su vez se ha ido haciendo más potente: las computadoras de sobremesa de hoy día tienen la potencia que poseían los mainframes hace algunos años. La segunda área es la de las comunicaciones; avances tales como los sistemas de comunicación vía satélite, los sistemas de teléfonos digitales y la Internet como tejido de las vidas de los seres humanos, han generando así un nuevo modelo de organización empresarial. Significa que ahora es posible conectar económica y eficientemente diferentes sistemas informáticos separados físicamente [Pressman, 2002]. Generando la nueva Sociedad Real que ofrece un medio diferente de comunicación, búsqueda de información, interacción y organización social, el ciberespacio [ETECSA, 2009].

Al mismo tiempo, el desarrollo de las grandes industrias tiene marcada influencia en el futuro económico de los países, determinando en muchos casos el posicionamiento en el mercado de sus economías. De aquí surge la importancia de contar con empresas competitivas y eficientes que potencien la actividad productiva para alcanzar elevados niveles de calidad [Osmany Jorge, 2007]. Es por ello que se requiere con gran premura y exactitud la información para una correcta toma de decisiones. Si a este planteamiento adicionamos el gasto de materiales y materias primas por ineficiencias de los procesos productivos industriales, se presenta el panorama donde la informatización debe entrar a jugar un papel preponderante.

Esto ha llevado a nuevos paradigmas y retos en el desarrollo de software donde su organización es clave para llevar a feliz término las necesidades de las empresas, existiendo una especialización en el desarrollo de aplicaciones informáticas, ya sean productos ERP, SCADA, BioInformáticos o SIG. Estos últimos dedicados a la representación de información geoespacial, y a la gestión de sus datos [Dr. José Batista Silva, 2005]; que principalmente en plataforma de escritorio, se han arraigado a un gran número de negocios en los cuales se maneja localización geográfica, convirtiendo el dato geográfico en fuente de información vital para la toma de decisiones. El número de negocios que se interesan por incluir funcionalidades de representación geográfica es muy variado y creciente [msdn, 2010].

Teniendo en cuenta el desarrollo de las comunicaciones, la competitividad de las empresas, los nuevos esquemas productivos de desarrollo y la creciente demanda de los Sistemas de Información Geográfica, surge como una necesidad garantizar una correcta comunicación entre los SIG y aplicaciones externas que se encuentran físicamente separadas, permitiéndose la manipulación y el intercambio de información entre los mismos, sobre la base de estándares abiertos.

En el centro de desarrollo GEYSED de la Universidad de las Ciencias Informáticas existen proyectos productivos, cuyo objetivo es lograr la implementación de un Sistema de Información Geográfica en plataforma de escritorio a través de la herramienta Quantum GIS (QGIS), dirigido a personalizar y satisfacer las necesidades de representación y análisis cartográfico. En el QGIS se presenta la misma necesidad de integración con soluciones existentes en ambientes de desarrollo heterogéneos.

A partir del análisis de la situación explicada con anterioridad se define como el **problema a resolver** que: *QGIS no permite la comunicación con aplicaciones externas*, como pudiera ser, el proyecto 171 que se desarrolla en la UCI. Para llevar a cabo la investigación es necesario estudiar la *Comunicación entre aplicaciones informáticas en plataforma de escritorio* que constituye el **objeto de estudio**, y como **objetivo general:** *Desarrollar funcionalidades a QGIS que le permitan comunicarse con un sistema informático externo específico* incidiendo directamente en los *Mecanismos de comunicación entre aplicaciones informáticas,* que representa el **campo de acción.**

Se propone como **Hipótesis:** *Si se incorporan a QGIS funcionalidades de comunicación con aplicaciones externas se permitirá minimizar el impacto de desarrollo en la integración de QGIS a diversos negocios ya existentes.*

Para cumplir con el objetivo trazado en esta investigación, es necesario llevar a cabo un conjunto de **tareas investigativas** que permitan, de manera sistemática y creciente avanzar en su desarrollo para lograr dar solución al problema científico que le dio origen a la misma:

1. Evaluar las tecnologías existentes de comunicación entre aplicaciones de escritorio en diversos Sistemas Operativos.
2. Caracterizar las técnicas de extensión de sistemas aplicables a QGIS.

3. Diseñar la posible solución mediante la tecnología de comunicación adecuada.
4. Desarrollar la solución.
5. Probar la solución.

Como consecuencia de las tareas y actividades desarrolladas en el proceso de investigación se esperan los siguientes resultados:

1. Documentación del estado del arte sobre tecnologías de comunicación de aplicaciones en plataforma de escritorio.
2. Documentos ingenieriles de diseño de la solución propuesta.
3. Desarrollo de funcionalidades de comunicación incorporadas a QGIS.
4. Documento de informe de la investigación.

Para la realización de estas tareas se utilizarán diferentes Métodos de Investigación:

Los Métodos Teóricos permiten estudiar las características del objeto de investigación que no son observables directamente, facilitan la construcción de modelos e hipótesis de investigación y crean las condiciones para ir más allá de las características fenomenológicas y superficiales de la realidad, contribuyendo al desarrollo de las teorías científicas y para su ejecución se apoyan en el proceso de análisis y síntesis posibilitando el conocimiento del estado del arte del fenómeno [Martinto, 2009).

1. Histórico – lógico: estudia la trayectoria real de los fenómenos y acontecimientos en el decursar de su historia, se usa para el estudio de la evolución y desarrollo de los métodos de comunicación existentes como base para el desarrollo de la presente investigación.
2. Modelación: es justamente el proceso mediante el cual creamos modelos con vistas a investigar la realidad y es utilizado para representar de manera funcional y gráfica la herramienta que se propone.
3. Analítico - Sintético: es una operación intelectual que posibilita descomponer mentalmente un todo complejo en sus partes y cualidades, se utilizará para el análisis de teorías, documentos, y materiales, de manera que se procese la información y se elaboren conclusiones.

Capítulo 1: Fundamentación Teórica

El presente capítulo muestra una exposición de los conceptos y aspectos teóricos asociados al dominio del problema, que posibilitará una mejor comprensión del tema tratado. Además de hacer una investigación sobre el surgimiento y desarrollo de las tecnologías de comunicación entre procesos, exponiendo las principales características de las mismas y seleccionado la que se empleará en la resolución del problema.

1.1. Conceptos asociados al dominio del problema

Para la comprensión correcta del trabajo de tesis, es necesario especificar el significado de algunos conceptos básico que acompañarán la investigación, siendo de esta forma, conducentes y esenciales objetivos para lograr la elaboración de la estrategia definitiva.

Primeramente se debe conocer que una *aplicación informática* es un tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajo, además que suele resultar una solución informática para la automatización de algunas tareas complicadas [Garcerant, 2008). Un ejemplo de aplicaciones son los *Sistemas de Información Geográfica* (en lo siguiente SIG) que van a hacer los programas que permiten capturar, manejar, manipular, analizar y gestionar información de tipo cartográfico, gráfico y alfanumérico, generando las denominadas relaciones topológicas entre variables espaciales, con el objetivo de resolver problemas de gestión y planificación. Técnicamente se puede definir un SIG como una tecnología de manejo de información geográfica formada por equipos electrónicos (hardware) programados adecuadamente (software) que permiten manejar una serie de datos espaciales (información geográfica, datos geográficos) y realizar análisis complejos con estos siguiendo los criterios impuestos por el equipo científico (personal o equipo humano) [msdn, 2010).

En una arquitectura de comunicación cliente servidor, el *emisor* (del latín *emissor*) que es aquello que manifiesta una opinión o noticia, enuncia un mensaje en un acto comunicativo, en fin es el dispositivo que transmite los datos, donde el *mensaje* lo conforman los datos a ser transmitidos [Humboldt, 2010). Este mensaje realiza un recorrido desde el origen hasta el destino a través del *medio* y es capturado por el

receptor que es el dispositivo de destino de los datos, es quien está en condiciones de procesarlos e interpretarlos.

La *comunicación de datos entre aplicaciones informáticas* es el proceso de transmitir y recibir información entre objetos que se ejecuten en distintos ámbitos (ya sea en un mismo equipo o en Internet) y los conceptos de *Emisor*, *Mensaje*, *Medio* y *Receptor* son básicos e indispensables [Pleglase, 2006). También se pone de manifiesto el *proceso de transmisión de mensajes* que va a permitir el envío de mensajes entre distintos puntos de una red teleinformática mediante el uso de una o más computadoras que efectúan la recepción de los mismos, los almacenan y luego se encargan de retransmitirlos a los puntos de la red, para los que estaban destinados, se pueden enviar mensajes de diversas longitudes y distintos formatos en dependencia de la vía que se utilice aquí es donde participa el *protocolo*, que su papel es el de llevar un lenguaje común con el que se realiza la transmisión y recepción para poder entenderse las aplicaciones y va a envolver toda la comunicación .

Las *transmisiones* de datos pueden ser *Síncrona* o *Asíncrona*. La primera es una técnica que consiste en el envío de un conjunto de caracteres que configura un bloque de información comenzando con un conjunto de dígitos de sincronismo y terminando con otro conjunto de dígitos de final de bloque, dicha transmisión se realiza con un ritmo que se genera centralizadamente en la red y es el mismo para el emisor como para el receptor, mientras que la segunda se da lugar cuando el proceso de sincronización entre emisor y receptor se realiza en cada palabra de código transmitido, envía señales de arranque y parada al principio y al final que permiten avisar al receptor de que está llegando un dato y le brinda suficiente tiempo al receptor de realizar funciones de sincronismo antes de que llegue la siguiente señal, además no existe ninguna relación temporal entre la estación que transmite y la que recibe, es decir, el ritmo de presentación de la información al destino no tiene por qué coincidir con el ritmo de presentación de la información por la fuente. Es un tipo de relación típica para la transmisión de datos. En este tipo de red el receptor no sabe con precisión cuando recibirá un mensaje.

1.2. Objeto de estudio

1.1.1. Descripción General

La necesidad de comunicación entre los sistemas se ha abordado cabalmente desde el punto de vista software En el presente trabajo se definen las distintas etapas para conseguir que una simple conexión entre dos aplicaciones se convierta en un sistema de comunicación completo. La realidad es que ningún sistema diseñado para la comunicación a través de una red tele-informática puede ocultar completamente el hecho de la existencia de la misma y las complejidades que esta conlleva [Pleglase, 2006). En la situación actual no existe realmente un sistema de comunicación que cubra todas las etapas, siendo lo habitual que estos sistemas lleguen únicamente hasta un nivel de transporte, es decir, al mero enrutamiento de mensajes a través de una red tele-informática [Pleglase, 2006).

El resto de las etapas necesarias para obtener una comunicación efectiva se suele dejar en manos de los programadores de aplicaciones. Para simplificar la labor de estos se han realizado distintos esfuerzos para construir herramientas software que simplifiquen la tarea.

En la actualidad existen diversas técnicas para establecer comunicación entre diferentes aplicaciones, están divididas dentro de métodos para: paso de mensajes, sincronización, memoria compartida y llamadas de procedimientos remotos (RPC). Para seleccionar un método de comunicación entre procesos (IPC, *Inter-process Communication* en inglés) se tiene en cuenta que puede variar dependiendo del ancho de banda y latencia (el tiempo desde el pedido de información y el comienzo del envió de la misma) de la comunicación entre procesos y del tipo de datos que están siendo comunicados. Su objetivo es la ocultación de la problemática asociada a la comunicación entre aplicaciones de forma que la construcción del software no se vea afectada por este hecho, minimizando el impacto de desarrollo en la integración de las mismas con los diversos negocios ya existentes.

1.1.2. Situación Problemática

A partir del surgimiento de Internet en el año 1983, con la adopción del protocolo TCP/IP, los avances y los logros alcanzados en la rama de la informática crecen vertiginosamente. En un ordenador, computadora o sistema de cómputo, el software representa las instrucciones responsables para que el hardware (la máquina) realice su tarea.

En la sociedad actual la computadora se ha tornado imprescindible, sus usos y aplicaciones se han extendido prácticamente a todas las esferas del ser humano, donde el desarrollo de las organizaciones demanda una enorme cantidad de información, obligando a las empresas a tomar decisiones cada vez más precisas y con mayor rapidez. Es por ello que las aplicaciones informáticas juegan un papel protagónico pues ayudan al desarrollo de las diferentes actividades, automatizando una gran cantidad de información para facilitar su acceso y manipulación. En ocasiones existe la solidaridad de documentación entre ellos, pero no siempre tienen la vía para comunicarse carecen de un componente que se los permita, imposibilitando de esta forma que compartan datos importantes. Para ello se crean software intermediarios que tengan la funcionalidad de comunicar aplicaciones que necesiten intercambiar información, pues no siempre es conveniente agregar nuevas soluciones informáticas a los ya existentes, y si le damos la responsabilidad a otro, de este modo minimizamos el impacto de desarrollo en la integración.

En la actualidad existen aplicaciones de escritorio que permiten la representación geográfica, en los años 1960 y 1970 emergieron nuevas tendencias en la forma de utilizar los mapas para la valoración de recursos y planificación [Humboldt, 2010), surgen así en 1962 los SIG, los cuales tienen la capacidad de manipular estos datos espaciales, posibilitando visualizar y analizar la información de forma versátil e intuitiva, agilizando la tan importante toma de decisiones, dando respuesta a muchos problemas que requieren acceso a varios tipos de información que sólo pueden ser relacionadas por geografía o distribución espacial con esta tecnología se puede almacenar y manipular información usando geografía, analizar patrones, relaciones, y tendencias en la información, todo con el interés de contribuir a la toma de mejores decisiones.

El Quantum GIS es un SIG que nació oficialmente en mayo de 2002 y ofrece un número cada vez mayor de capacidades proporcionadas por las funciones básicas y compuestas desarrolladas en gran parte por un grupo de voluntarios, que con el tiempo han construido una gran base de código, valioso y útil, libre de usar y mejorar para todos. QGIS se puede adaptar a nuestras necesidades especiales con la arquitectura extensible de complementos, proporcionando bibliotecas que se pueden usar para crearlos, ofrece un creciente número de plugins (viene del inglés “plug in” o “enchufar) que no son más que pequeños archivos que añaden mayores funcionalidades y características y en su mayoría son aportados por la comunidad (Encyclopedia Beta,2003).

Se pueden instalar fácilmente usando el instalador de complementos de python, permiten personalizar la funcionalidad de una aplicación, pueden ser usados para posibilitarle a los desarrolladores de terceras partes crear las capacidades que se extienden a la misma, además para apoyar fácilmente añadiendo nuevas características y en ocasiones reducir el tamaño de un software e incluso para separar el código fuente de una aplicación, ya sea por la incompatibilidad de licencias de software (Nokia, 2011). Los plugins tienen gran importancia pues brindan la posibilidad de activarlos y desactivarlos cuando se quiera utilizar de esta forma la aplicación puede ser más rápida, además pueden ser distribuidos como una entidad única que los usuarios finales pueden ejecutar sin la necesidad de medidas adicionales de instalación.

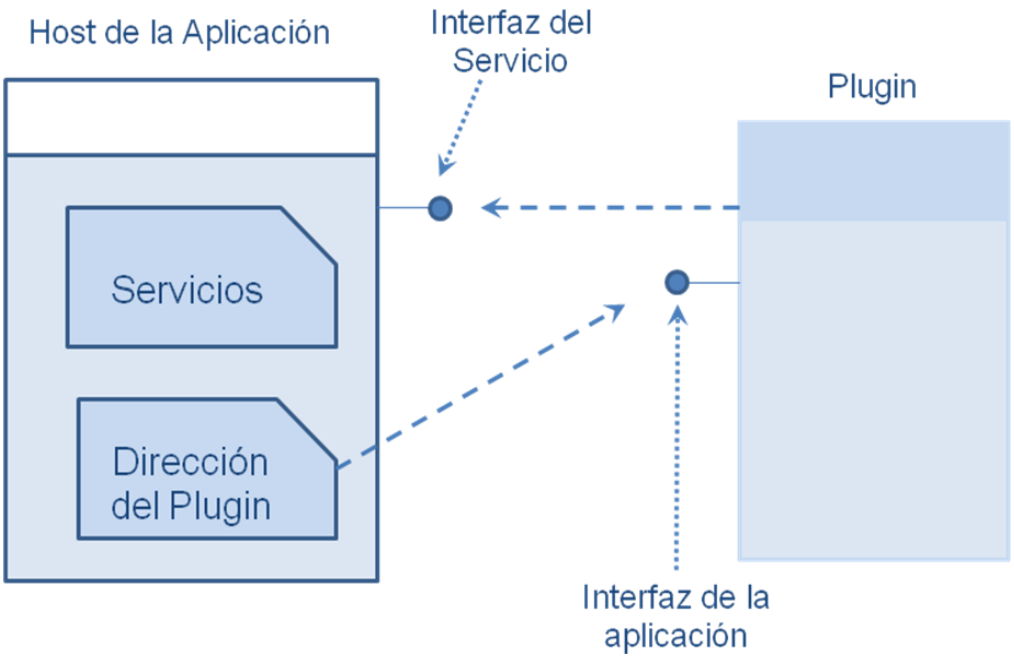


Figura 1: Ejemplo de Plugins

El Quantum GIS es muy usado por diferentes entidades [Gary E. Sherman, 2004 - 2007) y como ya se mencionaba consta de varios de estos complementos, pero en la actualidad no posee ninguno que le facilite la comunicación con una aplicación externa.

1.3. Análisis de tecnologías que permiten la comunicación entre aplicaciones.

1.3.1. Solución basada en RPC (*Remote Procedure Call* en inglés/ llamada a un procedimiento remoto)

Las primeras implementaciones de este mecanismo tuvieron lugar en 1976 y se comenzaron a utilizar a partir del año 1981 por parte de Xerox. Permite que los programas realicen llamadas a funciones localizadas en otras máquinas. Con este mecanismo los programadores no se tienen que preocupar por los detalles de la programación de la red.

Desde el punto de vista de un programador la llamada a una función remota es y funciona de la misma manera que lo haría si la llamada fuese local, en este sentido, se logra transparencia. La idea básica consiste en realizar llamadas a procedimientos

residentes en otros entornos como si fueran llamadas a procedimientos locales al programa.

Cada función pasa a tener dos partes: cliente, la máquina local donde se implementa la interface (prototipo de una función) para invocar las funciones remotas. Servidor, implementación de las funciones propiamente dichas.

RPC tiene gran apoyo para la conversión automática de datos y de las comunicaciones con otros sistemas operativos. Las comunicaciones RPC tienen como desventaja que se basan en la idea que el receptor está operativo para poder invocar una cierta función, el receptor siempre estará operativo y esperando a comunicarse. Esta comunicación la hace a través de eventos enviados por la red constantemente, si el paquete es muy grande y en una red local, puede traer retrasos en la misma (Manzano, 2009).

En cualquier entorno de hoy día las comunicaciones entre aplicaciones establecidas mediante RPCs, no son universales y existen soluciones específicas dependiendo del entorno de programación, sistema operativo y lenguaje utilizado, algunos ejemplos son:

- En Java, se utiliza un mecanismo conocido como RMI (Remote Method Invocation).
- En la plataforma .NET, se utiliza .NET Remoting (un mecanismo similar a RMI).

La primera invocación remota de métodos (RMI), es un mecanismo de expansión de RPC, orientada específicamente al lenguaje Java. Este lenguaje tiene una serie de peculiaridades que explican esta necesidad, especialmente el tratarse de un lenguaje orientado a objetos, interpretado y con recolector de basura incorporado. El hecho de ser un lenguaje orientado a objetos hace que no se exporte procedimientos sino métodos, además la forma natural de sobrecargar estos métodos es mediante la extensión de una clase básica, en este caso la clase “java.rmi.server” (Garcerant, 2008). La idea es tener objetos distribuidos. Para acceder al estado de un objeto sólo se realiza a través de métodos definidos por un objeto interface. Un objeto ofrece múltiples interfaces y una interfaz puede ser implementada por múltiples objetos.

Y la segunda permite crear fácilmente aplicaciones ampliamente distribuidas. Se puede construir aplicaciones cliente que utilizan los objetos en otros procesos en el mismo

equipo o en cualquier otro equipo que es accesible a través de su red. También se utiliza .NET Remoting para comunicar otros dominios de aplicación en el mismo proceso y tiene gran capacidad para poder trabajar desde una máquina con los objetos en memoria de la máquina Remota. Es flexible y fácilmente personalizable. Se puede reemplazar un protocolo de comunicación con otro, o un formato de serialización con otro sin tener que recompilar el cliente o el servidor. Puede comunicarse desde una aplicación web, una aplicación de consola, un servicio de Windows de casi cualquier cosa que desee utilizar. Cualquier aplicación puede alojar objetos remotos y ofrecer sus servicios a cualquier cliente en su computadora o red.

1.3.2. Solución basada en The Internet Communications Engine (ICE)

El Internet Communication Engine (ICE) es un estándar desarrollado por ZeroC [1]para el desarrollo de aplicaciones basada en objetos distribuidos, surge como alternativa del Common Object Request Broker Architecture (CORBA). ICE es un middleware ligero, abierto y orientado a objetos, es decir, proporciona herramientas APIs y soporte de bibliotecas para construir aplicaciones cliente-servidor orientadas a objetos. Una aplicación ICE se puede usar en entornos heterogéneos: los clientes y los servidores pueden escribirse en diferentes lenguajes de programación (C, C++, Java, C#, Visual Basic, Phyton, PHP), pueden ejecutarse en distintos sistemas operativos (Windows, Linux, y otras propietarias) y en distintas arquitecturas, y pueden comunicarse empleando diferentes tecnologías de red. Además, el código fuente de estas aplicaciones puede portarse de manera independiente al entorno de desarrollo (ZeroC, Inc, 2010).

El núcleo de ICE proporciona una sofisticada plataforma cliente-servidor para el desarrollo de aplicaciones distribuidas. También maneja una serie de servicios que gestionan la activación de servidores bajo demanda, la distribución de proxies a los clientes, la distribución de eventos asíncronos, la configuración de aplicaciones, etc. Manteniendo estos servicios disponibles como parte de la plataforma permite al desarrollador centrarse en el desarrollo de la aplicación en lugar tener que ocuparse de construir la infraestructura necesaria.

[1] Empresa con sede en Palm Beach Gardens, Florida, EE.UU., en torno al desarrollo y licenciamiento Internet Communications Engine, o ICE.

ICE permite que tanto los clientes como los servidores se comuniquen de forma segura a través de un cortafuego sin comprometer la seguridad. El tráfico entre el cliente y el servidor queda completamente encriptado utilizando certificados de clave pública y es bidireccional

1.3.3. Solución basada en Socket

Las primeras implementaciones del mecanismo de comunicación de sockets tuvieron lugar en la universidad de Berkeley sobre el sistema operativo Unix BSD. Esta implementación estuvo motivada por la evidente necesidad de que los programadores dispusieran de una interfaz programática que les permitiera hacer uso de la red para comunicar aplicaciones. La idea básica inicial que hay detrás de esta librería es establecer un camino extremo a extremo entre dos aplicaciones y de hecho el nombre de socket (enchufe) viene a significar esto “enchufar un cable a la aplicación” (Encyclopedia Beta, 2003). Es un fichero existente en la máquina cliente y en la máquina servidora, en el lado del cliente se utiliza la clase Socket y en el del servidor el Server Socket, que sirve en última instancia para que el programa servidor y el cliente lean y escriban la información. Esta información será la transmitida por las diferentes capas de red.

Los sockets se crean y se utilizan con un sistema de peticiones o de llamadas de función a veces llamados interfaz de programación de aplicación de sockets (API, application programming interface), para los cuales se necesita un protocolo de comunicación que permita el intercambio de octetos, una dirección del Protocolo de Red (Dirección IP, si se utiliza el Protocolo TCP/IP, aunque también es posible utilizar UDP o IPX), que identifica una computadora y un número de puerto, que identifica a un programa dentro de una computadora.

El uso de la librería de socket como mecanismo de comunicación tiene bastantes ventajas, la primera y más evidente de las cuales es que es un mecanismo universal, sirve en cualquier entorno y circunstancia independientemente del sistema operativo, lenguaje de programación utilizado e incluso versión ya que la implementación es estable desde hace mucho tiempo (Manzano, 2009).

Tiene también otra ventaja y es la posibilidad de interactuar directamente con la capa de TCP/IP lo que permite ajustar cualquier facilidad que exista dependientemente de las necesidades y tener un control lo más completo posible de la comunicación y de las situaciones que se pueden dar en la misma.

Sin embargo estas ventajas se pueden convertir fácilmente en desventajas. La posibilidad de actuar directamente con la capa de TCP/IP complica la programación al obligar a conocer cómo funciona el mecanismo de comunicación, un ejemplo típico de esta situación es el desconocimiento por parte de los programadores noveles de que una lectura del socket no se corresponde con la lectura de un mensaje de aplicación sino de una trama de comunicación y que pueden ser necesarias varias lecturas del mismo para obtener la trama enviada. Además de estos problemas hay que evidenciar otro tipo de situaciones no muy ventajosas, así hay que denunciar que mediante este mecanismo no se puede asegurar que el canal está establecido punto a punto durante toda la duración de la sesión de comunicación, siendo habitual que un extremo siga actuando como si el canal estuviera establecido a pesar de que el otro extremo haya desaparecido. Tampoco se garantiza que los mensajes de aplicación lleguen completos al otro extremo ya que lo que se garantiza son las tramas de comunicaciones, el control de esta situación queda en manos del programador y obliga a la re-inicialización del mecanismo de comunicación.

1.3.4. SOAP (Simple Object Access Protocol)

SOAP (Protocolo Simple de Acceso a Objetos) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML, entre diversas aplicaciones Web desarrolladas bajo tecnologías diferentes, tomando como base protocolos ya establecidos y con gran aceptación en Internet, como HTML(para la transmisión) y XML(para la codificación de datos). Se adhiere al paradigma de la programación orientada a objetos. Además SOAP especifica el formato de mensaje que accede e invoca a los objetos (Garcerant, 2008).

La especificación SOAP menciona que las aplicaciones deben ser independientes del lenguaje de desarrollo, por lo que las aplicaciones cliente y servidor pueden estar escritas con HTML, DHTML, Java, Visual Basic u otras herramientas y lenguajes disponibles. Un mensaje SOAP tiene una envoltura, un encabezado y el cuerpo,

envelope (envoltura): es el elemento raíz del mensaje para describir su contenido y la forma de procesarlo; *header* (encabezado): es la información de identificación del contenido, un grupo de reglas de codificación para expresar las instancias de tipos de datos definidos por la aplicación; *body* (cuerpo): es el contenido del mensaje, una convención para representar las llamadas y las respuestas a procedimientos remotos.

Entre las ventajas que tiene el SOAP están (Garcerant, 2008):

- Es sencillo de implementar, probar y usar.
- Atraviesa "firewalls" y routers, pues estos "piensan" que es una comunicación HTTP.
- Tanto los datos como las funciones se describen en XML, lo que permite que el protocolo no sólo sea más fácil de utilizar sino que también sea muy sólido.
- Es independiente del sistema operativo y procesador.
- Facilidad para utilizar cualquier lenguaje: Los desarrolladores involucrados en nuevos proyectos pueden elegir desarrollar con el último y mejor lenguaje de programación que exista. SOAP no especifica una API, por lo que la implementación de la API se deja al lenguaje de programación, como en Java, y la plataforma como Microsoft .Net.
- No está atado a ninguna infraestructura de objeto distribuido: La mayoría de los sistemas de objetos distribuidos se pueden extender, y alguno de ellos admiten SOAP.
- Permite la interoperabilidad entre múltiples entornos: SOAP se desarrolló sobre los estándares existentes de la industria, por lo que las aplicaciones que se ejecuten en plataformas con dichos estándares pueden comunicarse mediante mensaje SOAP con aplicaciones que se ejecuten en otras plataformas.

Entre las desventajas de SOAP se tiene que:

- Las desventajas de la utilización de SOAP recaen en la dificultad para entender las especificaciones del protocolo, pues es un complejo esquema de codificación en el cual es necesario precisar que todos los mensajes se incluyan en un sobre, con el contenido del mensaje dentro de un elemento de cuerpo para que puedan ser entendidos por cada una de las aplicaciones Web que procesan el mensaje.

- SOAP convierte en opcionales elementos como encabezados y ofrece un amplio margen con respecto a lo que se puede incluir en el elemento de cuerpo y además cambia los nombres de métodos en etiquetas secundarias del cuerpo y los argumentos en etiquetas secundarias del nombre del método, lo que puede generar ciertos problemas de interoperabilidad.

1.4. Conclusiones

A partir del desarrollo del Capítulo 1 del presente trabajo comprendieron los conceptos básicos asociados al dominio del problema de la investigación. Se hizo un análisis de la situación problemática, además se realizó un análisis de las soluciones de comunicación entre aplicaciones existentes que sirve como punto de partida para continuar el estudio, para enriquecer los conocimientos sobre las tecnologías de comunicación existente (ver Anexo I).

Capítulo 2: Tendencias y tecnologías actuales a utilizar.

En este capítulo se detallan las tecnológicas a utilizar para la construcción de la herramienta propuesta, que incluye: tecnología de comunicación, herramientas de modelado, metodologías de desarrollo de software y lenguajes de programación. Se describen a continuación las principales ventajas que facilitaron su selección, además fueron comparadas y evaluadas de tal forma que se consideran factibles para la construcción de la aplicación propuesta pues son concurrentes con el ambiente de desarrollo del Quantum GIS.

2.1. Arquitectura de Comunicación

Al hablar de comunicación entre ordenadores resultan fundamentales dos conceptos: Protocolos y Arquitectura de comunicación. El primero es necesario debido a la complejidad que requiere la comunicación entre dos entidades de diferentes sistemas [Sergio Oyarce, 2008). El segundo es una estructura que organiza los protocolos con el objetivo de simplificar su diseño y generalmente se compone de un conjunto de capas o niveles.

Comunicación Punto a Punto (P2P)

Para establecer la comunicación entre procesos que se encuentren en diferentes computadoras es necesario seguir las instrucciones de una Arquitectura que guíe todo el transcurso de la integración. La Arquitectura de comunicación Cliente-Servidor es una de las más usadas en la actualidad. Los términos cliente y servidor no están directamente asociados a dos partes distintas de una aplicación, sino que más bien hacen referencia a los roles que las diferentes partes de una aplicación pueden asumir durante una petición. Por ejemplo los clientes emiten solicitudes de servicio a un servidor; los servidores proporcionan un servicio en respuesta a las solicitudes de los clientes. Sin embargo, la comunicación P2P permite el intercambio directo de información (Garcerant, 2008). Garantiza que los dos procesos que se encuentren interconectados asuman un mismo nivel de responsabilidades. Permite que los dos nodos se comporten como iguales entre sí, es decir, actúen simultáneamente como clientes y servidores; dando una atención única y recíproca.

2.2. Tecnología de comunicación

Para lograr una comunicación rápida y eficiente entre aplicaciones externas se necesita un software que garantice la conectividad entre todos sus componentes y la distribución de sus datos. Se precisa de un software que forme un conjunto de programas distribuidos, sencillos, consistentes e integrados que faciliten las tareas de programación y gestión de aplicaciones distribuidas. Por esta razón se propone una arquitectura de software para otorgar a la plataforma un conjunto extensible de servicios que facilitarán la implementación. Para darle solución se necesita la creación de una capa de software intermedio, que controle las comunicaciones entre contenidos y plataforma, que aporte algunas funcionalidades necesarias.

Internet Communications Engine (ICE)

Para resolver los problemas inherentes a sistemas heterogéneos y distribuidos, que dificultan la implementación de verdaderas aplicaciones, los proveedores de software ofrecen interfaces de programación y protocolos estándares tales como: RPC, .Net Remoting, Socket, SOAP y ICE. Estas han sido analizadas teniendo en cuenta diferentes indicadores, en los que se destaca la compatibilidad con el ambiente de desarrollo del QGIS y la usabilidad.

Finalmente se selecciona ICE porque es un Middleware[2] que ofrece de manera abstracta un soporte multiplataforma homogéneo, que facilita tener uniformidad en sus códigos.

ICE proporciona una serie de beneficios a los desarrolladores de aplicaciones (ZeroC, Inc, 2010):

- Mantiene una semántica orientada a objetos.
- Proporciona un manejo síncrono y asíncrono de mensajes.
- Soporta múltiples interfaces.
- Es independiente de la máquina.

[2] Es el software de conexión que consiste en un conjunto de servicios que permiten a múltiples procesos que se ejecutan en una o más máquinas de interactuar a través de una red.

- Es independiente del lenguaje de programación.
- Es independiente de la implementación, es decir, el cliente no necesita conocer cómo el servidor implementa sus objetos.
- Es independiente del sistema operativo.
- Soporta el manejo de hilos.
- Es independiente del protocolo de transporte empleado.
- Mantiene transparencia en lo que a localización y servicios se refiere.
- Es seguro.
- Permite hacer persistentes las aplicaciones.
- Tiene disponible el código fuente.

Fue publicada bajo los términos de la GNU General Public License.

2.3. Lenguajes de Programación (C++)

Un lenguaje de programación es una construcción mental del ser humano para formular programas de computadoras a través de un conjunto de instrucciones y obtener un determinado resultado que permite controlar el comportamiento de una máquina. Se componen de un conjunto de símbolos utilizables, términos con sentido único y reglas sintácticas y semánticas que permiten expresar instrucciones que luego serán interpretadas (Nokia, 2011). Existen diferentes lenguajes de programación y para la selección del mismo, que se empleará en el desarrollo de la propuesta de solución, es necesario tener en cuenta la compatibilidad con el ambiente de desarrollo del Quamtun GIS y la tecnología de comunicación seleccionada. Para que cumpla con este parámetro de entendimiento (que es fundamental) las opciones se resumen en Python y C++. El primero no ofrece seguridad en el código, pues el usuario tiene acceso al mismo, y se corre el riesgo que lo pueda modificarlo, no existe forma que se pueda bloquear. Por otra parte C++, a pesar de ser el lenguaje en que está implementado QGIS, ofrece mayor seguridad sobre el código, el cliente no tiene acceso y de esta forma no puede hacer modificaciones, manteniendo así la originalidad. Además está estandarizado y un mismo código fuente se puede compilar en diversas plataformas. C++ es uno de los lenguajes más rápidos en cuanto ejecución. Por todas las ventajas mencionadas es escogido para el desarrollo de la plataforma C++.

2.4. El Lenguaje Unificado de Modelado (UML) como soporte de la modelación de la solución propuesta

Unified Modeling Languaje (UML) es el lenguaje estándar de modelado para software, surge por la necesidad compartir fácilmente diseños gráficos realizados entre distintos diseñadores y servir de apoyo en los procesos de análisis de un problema. Permite a los desarrolladores visualizar, especificar, construir y documentar los resultados de su trabajo en esquemas ó diagramas estandarizados. Sus creadores pretendieron, con este lenguaje, unificar las experiencias acumuladas sobre técnicas de modelado e incorporar las mejores prácticas en un acercamiento estándar.

Entre las características fundamentales del lenguaje UML están:

- Permite modelar sistemas utilizando técnicas orientadas a objetos.
- Adecuado a las necesidades de conectividades actuales y futuras.
- Ampliamente utilizado por la industria del software.
- Reemplaza a decenas de notaciones empleadas por otros lenguajes.
- Modela estructuras complejas.
- Comportamiento del sistema: casos de usos, diagramas de secuencia, de colaboración, que sirve para evaluar el estado de las máquinas. [Jacobson I, 2000)

2.5. Metodología de desarrollo del Software

Una metodología para el desarrollo de software garantiza cumplir con los planes de producción del software dentro de una planificación y presupuesto establecido y alcanzar los objetivos del proyecto dándole solución a los requerimientos planteados por los clientes.

Entre las metodologías más reconocidas se encuentran el Proceso Unificado de Desarrollo (RUP), exponente de un proceso disciplinado sobre el desarrollo de software que empleado correctamente asegura la calidad de la aplicación que se produce en todas y cada una de sus fases de desarrollo.

El Proceso Unificado de Desarrollo de Software (RUP)

RUP constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Es un proceso bien definido, estructurado y adaptable a las características y necesidades de cada proyecto específico, abarca tanto pequeños como grandes proyectos.

Es un proceso iterativo e incremental que divide su ciclo de vida en cuatro fases, *Inicio:* se determina la visión y alcance del proyecto; *Elaboración:* se analiza el dominio del problema y se determina la arquitectura óptima; *Construcción:* se obtiene la capacidad operacional inicial; *Transición: s*e obtiene la liberación del proyecto, además de los nueve flujos de trabajo, de ellos seis de ingeniería y tres de soporte, de esta forma se garantiza una mejor organización para el desarrollo del software[Jacobson I, 2000).

RUP hace uso de arquitectura basada en componentes, permite la verificación de la calidad del software y el modelado visual, además, pretende implementar las mejores prácticas en Ingeniería de Software. RUP brinda un proceso integrado que utiliza el estándar de notación UML (del inglés Unified Modeling Language) para permitir desarrollar un proceso de forma iterativa e incremental a partir de la identificación e implementación de los casos de uso.

Entre las características que más se destacan del RUP están:

- *Dirigido por casos de uso*: Los casos de usos representan el hilo conductor que orienta las actividades de desarrollo, identifican cómo debería interactuar el sistema con el usuario. Se centra en la funcionalidad que el sistema debe poseer para satisfacer las necesidades del usuario.
- *Centrado en la arquitectura*: Establece una arquitectura que guíe el desarrollo del sistema y en la que el usuario y el equipo de trabajo deben estar de acuerdo, describe los modelos que son más importantes para su construcción.
- *Iterativo e incremental:* El desarrollo iterativo brinda la posibilidad de que los elementos sean integrados progresivamente, facilita el rehúso y resulta un producto más robusto pues los errores se van corrigiendo en cada iteración [Jacobson I, 2000).

2.8. Herramienta CASE para el modelado

Una Herramienta CASE (del inglés Computer Aided Software Engineering) es una aplicación informática destinada a aumentar la productividad en el desarrollo de software, reduciendo el coste de las mismas en términos de tiempo y de dinero. [Leidiana Martínez Diaz, 2010)

Entre sus principales funciones se encuentra aumentar el nivel de calidad del sistema construido y reducir el tiempo y coste de desarrollo. Además ayuda a la reutilización del software, portabilidad y estandarización de la documentación, gestionando de forma global todas las fases de desarrollo del software con una misma herramienta.

Visual Paradigm

Visual Paradigm es una herramienta CASE que utiliza UML como lenguaje de modelado. Soporta el ciclo de vida completo del desarrollo de software: Análisis y Diseño orientados a objetos, Construcción, Pruebas y Despliegue. Está diseñada para una amplia gama de usuarios interesados en construir sistemas fiables con el uso del paradigma orientado a objetos, incluyendo actividades como ingeniería de software, análisis de sistemas y análisis de negocios. [Jacobson I, 2000)

Esta herramienta multiplataforma es una tecnología libre, fácil de instalar y actualizar que se encuentra disponible en varios idiomas. Facilita la realización de los diagramas de modelados grandes y complejos que siguen el estándar de UML, debido a sus características gráficas y estructurales:

- Permite el uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Ofrece amplias características de modelado de casos de uso incluyendo la función completa de UML, diagrama de casos de uso y editor de flujos de eventos.
- Permite Diseñar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.
- El software de modelado UML ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor coste.

- La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

2.10. Entornos de Desarrollo Integrado (IDE)

El entorno de desarrollo integrado en inglés Integrated Development Environment (IDE), es un software informático compuesto por un conjunto de herramientas de programación, que puede ser multiplataforma, soportar diversos lenguajes de programación, importar y exportar proyectos, integrarse con Sistemas de Control de Versiones y Framework populares, compuesto por un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de forma amigable.

QT Creator

Qt es un conjunto de librerías multiplataforma diseñadas principalmente para desarrollar interfaces gráficas de usuario, aunque es posible desarrollar una aplicación completa haciendo sólo uso de éstas librerías.

Qt utiliza el lenguaje de programación C++ de forma nativa. Presenta toda la documentación de Qt y ejemplos a través del plugin de Ayuda Qt. Permite construir interfaces de usuario complejas de una forma visual y rápida, además de poseer auto-completado de código. QT es un proyecto de Código Abierto y gratuito de libre distribución. [Nokia, 2011)

2.11. Conclusiones.

En el presente capítulo se justificaron la elección de las herramientas y metodologías que serán utilizadas durante el desarrollo de la propuesta de solución, donde se decide desarrollar la aplicación en el lenguaje C++ utilizando el IDE Visual Studio siguiendo la metodología RUP. Los diagramas serán modelados en la herramienta CASE Visual Paradigm.

Capítulo 3: Presentación de la solución propuesta

El presente capítulo tiene como objetivo describir las características fundamentales del sistema. Se expone de forma detallada los requisitos funcionales y no funcionales a tener en cuenta y se presenta la solución desde la perspectiva del sistema. Se definen el modelo de dominio, diagrama de casos de uso del Sistema y actores del sistema con sus respectivas descripciones textuales, así como los diagramas de clases del diseño.

3.1. Solución propuesta

Para que se cumplan los objetivos del sistema es necesario resolver el problema que trae consigo el uso de plataformas heterogéneas, es decir, los componentes del sistema se ejecutan sobre distintos sistemas operativos o son desarrollados con distintos lenguajes de programación o ambos. Para ello se integrará el QGIS que permitirá distribuir los datos, establecer cómo los módulos de una aplicación se interrelacionan y operan coordinadamente posibilitando rebasar los obstáculos de comunicación existentes entre las distintas partes del sistema en función de obtener un sistema único.

Para un mejor manejo, el sistema QGIS ofrece una arquitectura extensible a través de complementos. Esta variante evita grandes cambios en el código del QGIS, es por ello que se implementará un plugin que permita acceder a las funcionalidades del mismo desde una aplicación externa. Una vez integrado el software externo con el SIG, el plugin brindará la posibilidad de acceder y administrar las funcionalidades del SIG, es decir, permite gestionar todo lo referente a los datos geoespaciales.

3.2. Modelo de Dominio

En el estudio realizado del problema planteado, no es posible identificar procesos del negocio, ya que está altamente centrado en tecnologías informáticas, haciéndose difícil determinar los elementos más importantes que intervienen, así como el establecimiento de las reglas de funcionamiento. Por estas razones se propone realizar un modelo conceptual o modelo de dominio.

El Modelo de Dominio captura los tipos más importantes de objetos que existen o los eventos que suceden en el entorno donde estará el sistema y no incluyen las

responsabilidades de las personas que ejecutan las actividades [Garcerant, 2008). Sirviendo de las ventajas que brindan los diagramas UML para la representación de conceptos, el Modelo de Dominio se representa en forma de diagrama de clases en el que se muestran los conceptos u objetos del dominio del sistema en cuestión, y las asociaciones entre ellos.

3.1.1. Representación del Modelo de Dominio

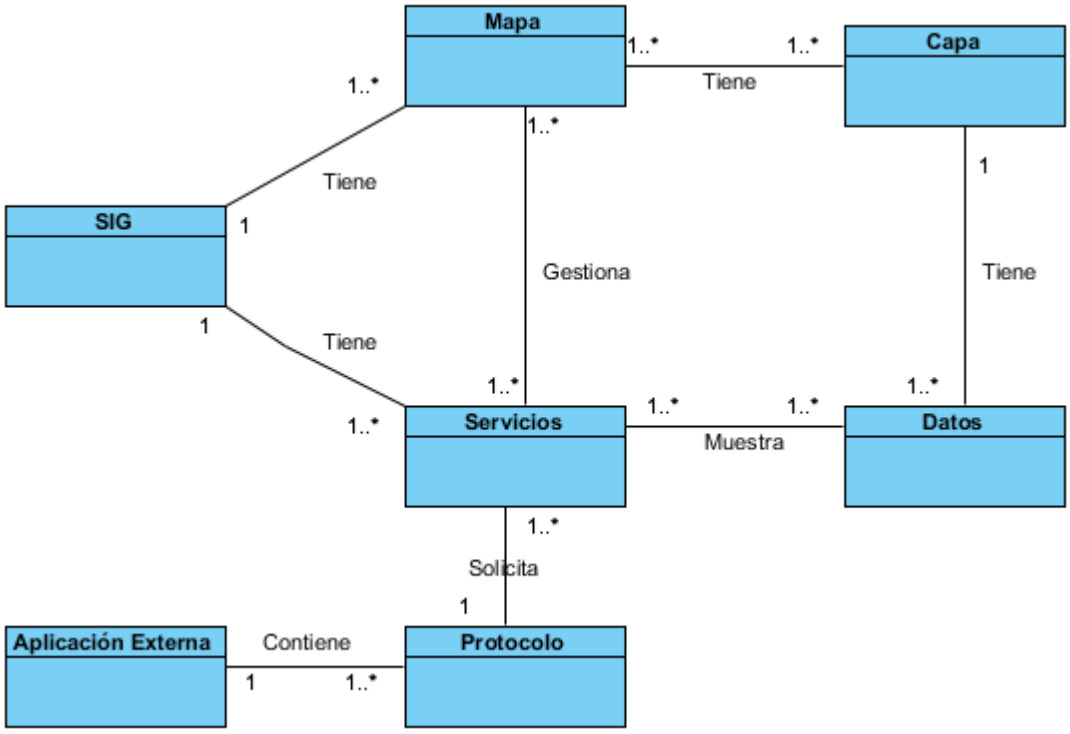


Figura 2: Modelo de dominio

3.1.2. Definición de las clases del Modelo de Dominio

El SIG brinda una serie de servicios solicitados por un software externo. Esta asistencia se manifiesta a través de la gestión de las capas que poseen los mapas, los cuales tienen toda la información socioeconómica, tratada en los distintos tipos de datos referentes a todos los sectores de la economía y la sociedad.

A continuación se explica en qué consiste cada una de las clases que conforman el modelo de dominio.

SIG: Software encargado de la gestión de datos geoespaciales.

Servicios: Son las funcionalidades requeridas por la aplicación externa.

Aplicación Externa: Software que necesita trabajar con información geográfica y para ello solicita los servicios de un SIG.

Protocolo: Es la vía o lenguaje de comunicación empleado entre las dos aplicaciones que se van a comunicar, a través de él se transmitirán los datos requeridos.

Mapa: Es una representación gráfica y métrica de una porción de territorio sobre una superficie bidimensional, generalmente plana, pero que puede ser también esférica como ocurre en los globos terráqueos.

Capa: Se utilizan para controlar la visibilidad de la geometría.

Datos: Descripción elemental o una información.

3.3. Requisitos funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. De esta forma se garantiza cumplir con los objetivos trazados y darle solución a todas las necesidades existentes. Como requisitos funcionales de la solución propuesta se tienen:

- **RF 1** Establecer comunicación: el sistema debe permitir establecer la comunicación entre el QGIS y la aplicación externa, permitiendo de esta forma que se puedan enviar y recibir información entre los diferentes nodos.
- **RF 2** Solicitar servicio Cargar Cartografía: el sistema de debe permitir cargar una cartografía.
- **RF 3** Realizar solicitud del servicio Zoom: el sistema de debe permitir realizar Zoom a la cartografía cargada. De esta manera le facilita la vista de todo el mapa al cliente.
- **RF 4** Dar órdenes de cambios: el sistema de debe permitir recibir información en la aplicación externa desde el QGIS.

3.4. Requisitos no funcionales

Los requisitos no funcionales representan propiedades o cualidades que el producto debe tener lo que hace del mismo que sea más atractivo, más usable, rápido y más confiable.

Fiabilidad

- Debe existir una comunicación eficiente y fiable, garantizando no existan pérdidas de información (eventos, paquetes, etc.).
- Debe proveerse, a los servicios, de mecanismos de tolerancia ante fallos que permitan recuperarse ante errores.

Software.

- La aplicación podrá ser ejecutada sobre cualquier sistema operativo.

Hardware.

- Los requisitos mínimos para la ejecución de la aplicación son: Procesador Intel Pentium IV de 2.5 MHz (o equivalente) y versiones posteriores, 512 de RAM y 600 MB de espacio en disco disponible.

Restricciones en el diseño y la implementación.

- El sistema será creado sobre la base de la Metodología Proceso de Desarrollo Unificado, usando la herramienta Visual Paradigm para el modelado de los artefactos que se obtendrán durante el desarrollo del software.
- El lenguaje de programación en el cual se desarrollará el middleware es C++.
- Tecnología middleware: Internet Communication Engine 3.3.1.
- Deberá ser desarrollado sobre Software libre.

Rendimiento

- Durante el intercambio de variables, debe existir una comunicación eficiente, garantizando la integración y una velocidad en la comunicación adecuada para aplicaciones de visualización y gestión de datos geoespaciales.
- Realizar implementaciones que permitan transacciones en Tiempo Real para procesos críticos.

Soporte

- El desarrollo del sistema orientado a la exposición de interfaces y servicios debe brindar alta interoperabilidad para que aplicaciones externas implementadas en diferentes lenguajes y sistemas operativos, puedan comunicarse de manera fácil y eficiente.
- El sistema debe ser implementado con tecnologías libres que permitan el desarrollo de aplicaciones orientada a servicios y que sean compatibles con todos los sistemas operativos existentes, buscando una solución robusta y universal.

Confiabilidad

- Se debe garantizar que la aplicación esté disponible las 24 horas del día.
- El sistema debe poder ser usado por cualquier persona que posea conocimientos básicos de computación (trabajo con ventanas, trabajo con menú, trabajo con el mouse).

Usabilidad

- El sistema debe permitir el acceso a los servicios de manera rápida y segura.

Portabilidad

- El middleware podrá ser usado por Quantum GIS y cualquier sistema externo.

3.5. Modelo de Casos de Usos del Sistema

El modelo de casos de uso (CU) describe la funcionalidad propuesta del nuevo software. Está formado por actores y casos de uso del sistema. Un caso de uso

constituye una técnica utilizada para describir el comportamiento del sistema, a través de un documento narrativo que define la secuencia de acciones que obtienen resultados de valor para un actor que utiliza un sistema para completar un proceso, sin importar los detalles de la implementación. Por su parte un actor representa a personas, otros sistemas o hardware externo que interactúa con el sistema.

3.5.1. Descripción de los Actores

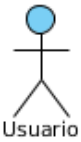
Actor	Descripción
	Usuario encargado de solicitar cargar una cartografía así como realizar un Zoom In a la misma.
	El actor QGIS se encarga de enviar órdenes cuando en él se ejecuta una nueva cartografía.
	El Usuario es una generalización de los actores Aplicación Cliente y QGIS e inicializa el la acción de Establecer Comunicación.

Tabla 1 Descripción de los actores del sistema

3.5.2. Diagrama de Casos de Usos del Sistema

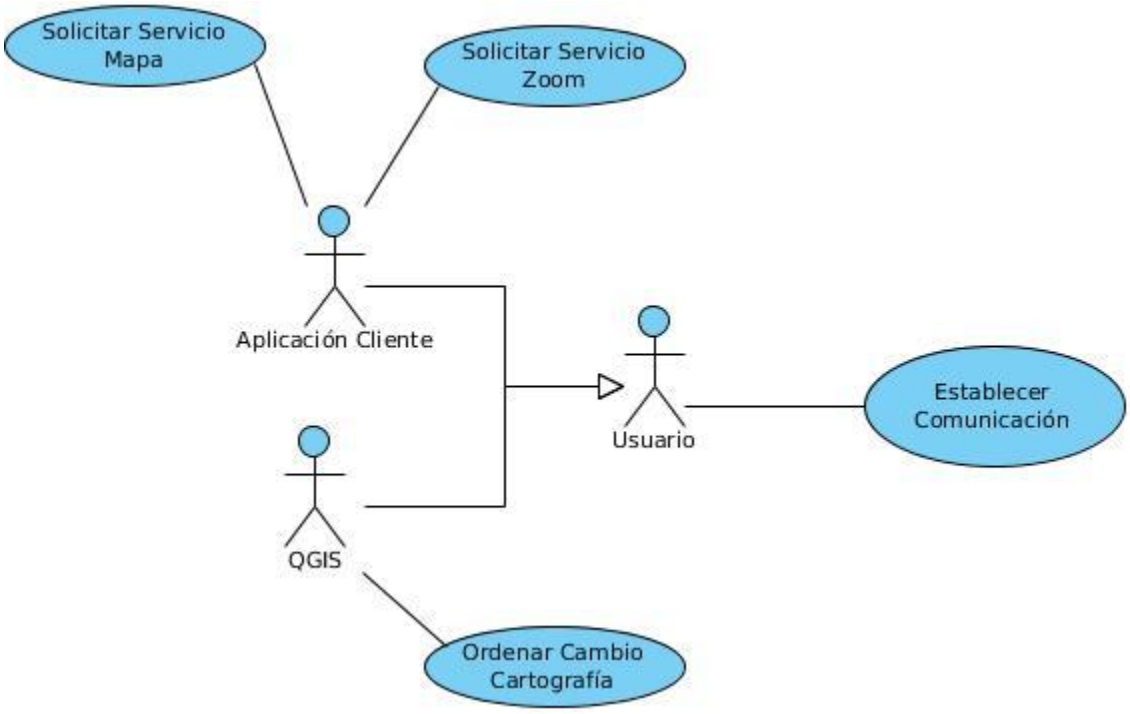


Figura 3 Diagrama de Casos de Usos del Sistema

3.5.3. Descripción textual de los Casos de Usos del Sistema

En el siguiente epígrafe se describen detalladamente los casos de uso presentes en el Diagrama de Casos de Uso del Sistema. Se situará el propósito general de cada caso de uso, el actor que le da inicio al mismo, así como las precondiciones y las poscondiciones para su funcionamiento.

Caso de Uso:	**Establecer comunicación**
Actores:	Usuario
Resumen:	El caso de uso inicia cuando el usuario necesita intercambiar Información.
Precondiciones:	Para realizar la comunicación debe existir integración con el QGIS.
Referencias	RF 1
Prioridad	Crítico
Flujo Normal de Eventos	

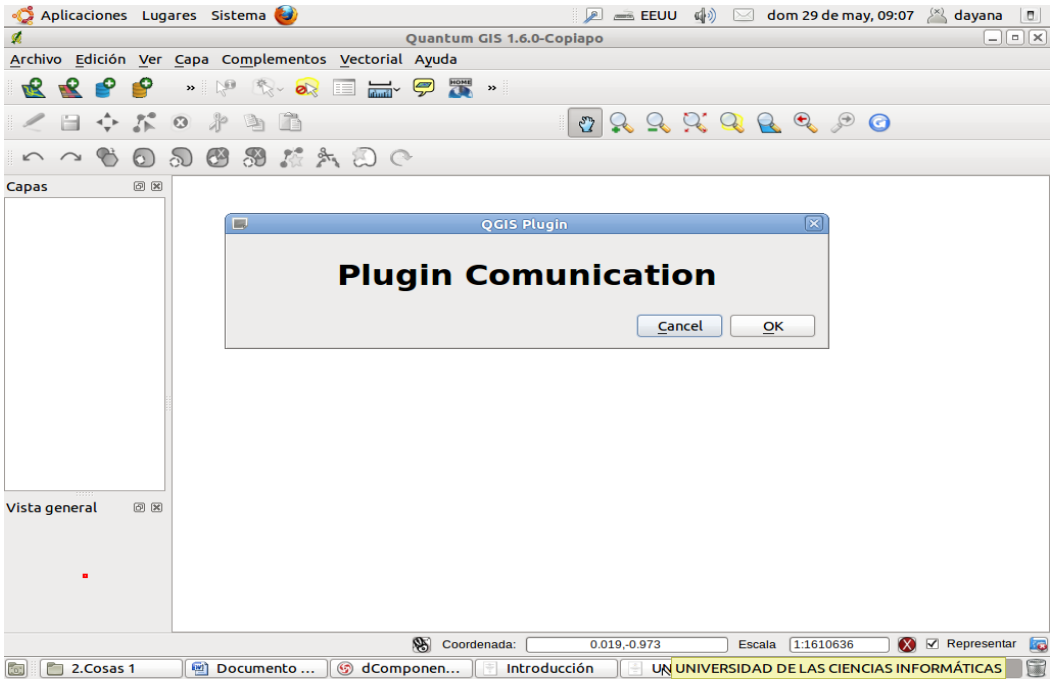
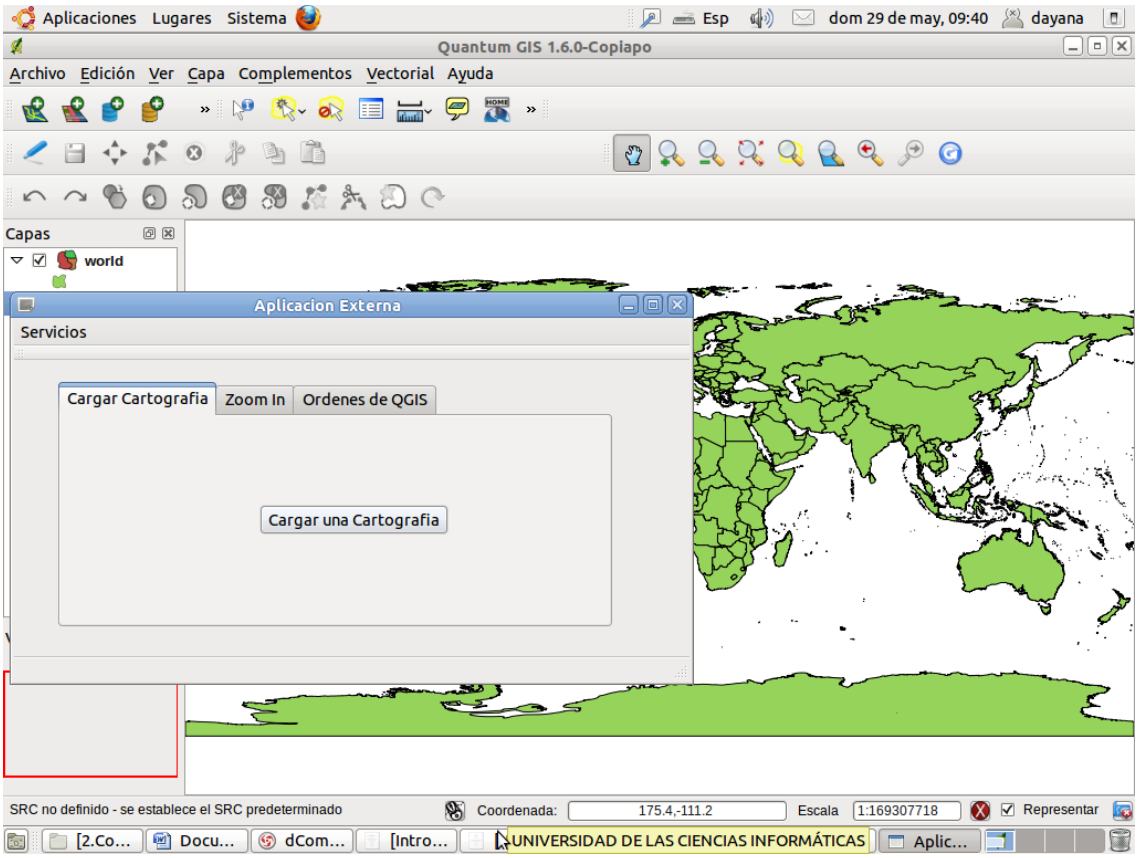
Acción del Actor	Respuesta del Sistema
1. El Usuario solicita la comunicación.	2. Establece la comunicación.
Prototipo de Interfaz	
	
Poscondiciones	Se obtienen los datos para establecer la comunicación.

Tabla 2 Descripción del CU Configurar comunicación

Caso de Uso:	**Solicitar servicio Cargar Cartografía**
Actores:	Aplicación Cliente
Resumen:	El caso de uso inicia cuando el cliente selecciona la opción de Cargar una cartografía.
Precondiciones:	Debe existir una comunicación entre la aplicación externa y el QGIS.
Referencias	RF 2
Prioridad	Secundario
Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema

	1-El sistema brinda la opción de cargar una cartografía.
2-El actor selecciona la opción Cargar cartografía.	3-El sistema procede a ejecutar permitiendo abrir o cargar una cartografía desde el QGIS.

Prototipo de Interfaz



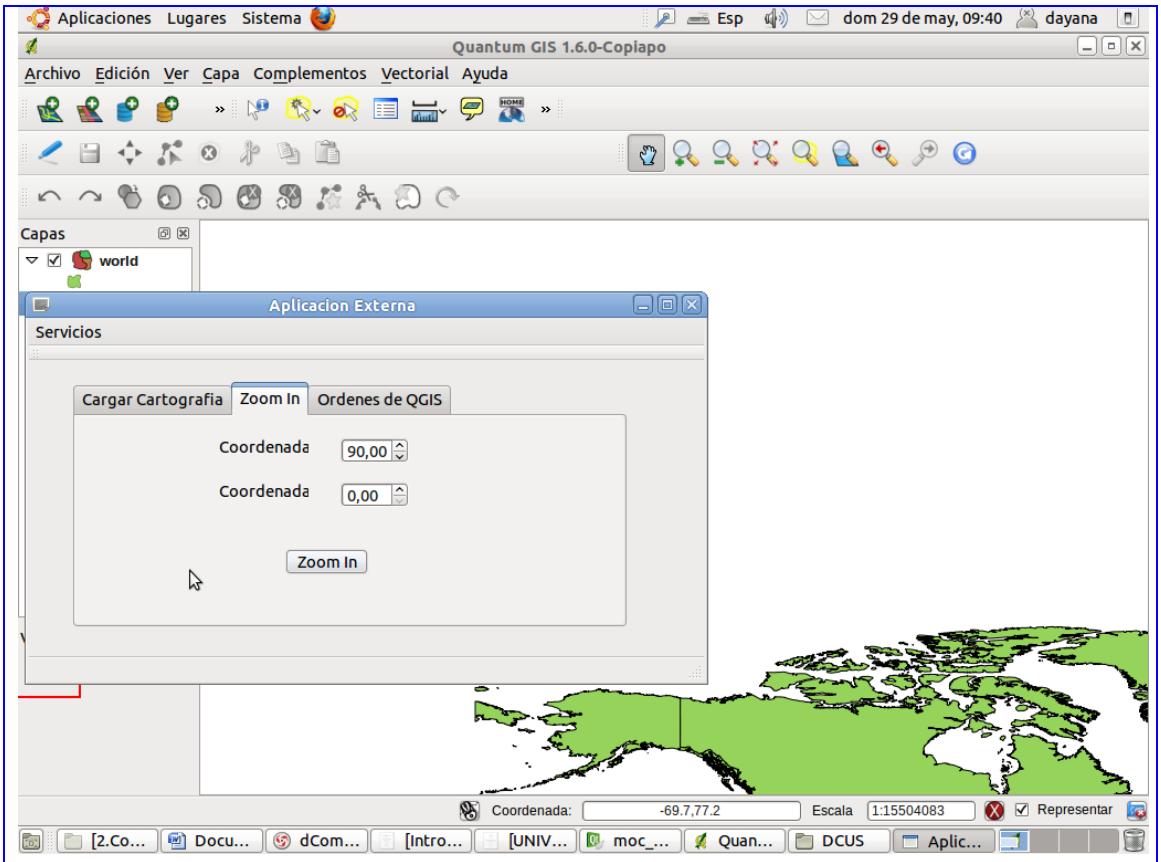
Poscondiciones	Se muestra el mapa desde el QGIS.

Tabla 3 Descripción del CU Solicitar servicio Cargar Cartografía

Caso de Uso:	**Realizar Solicitud Zoom**
Actores:	Aplicación Cliente
Resumen:	El caso de uso inicia cuando el cliente selecciona el botón y procede a realizar su función; la posición seleccionada será

	enviada hacia adelante logrando una mejor visión del elemento seleccionado.
Precondiciones:	Para lograr una navegación tiene que tener cargada o creada una capa y la misma tiene que estar seleccionada para poder trabajar en ella.
Referencias	RF 3
Prioridad	Secundario

Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema
	1- El sistema brinda la opción de acercar zoom.
2-El actor selecciona la opción de Zoom	
3-El cliente entra las coordenadas donde desea que se realice un Zoom.	3- El sistema captura la posición seleccionada y la misma será enviada hacia adelante realizando un aumento en el lienzo, logrando una mejor visión del elemento seleccionado.
Prototipo de Interfaz	



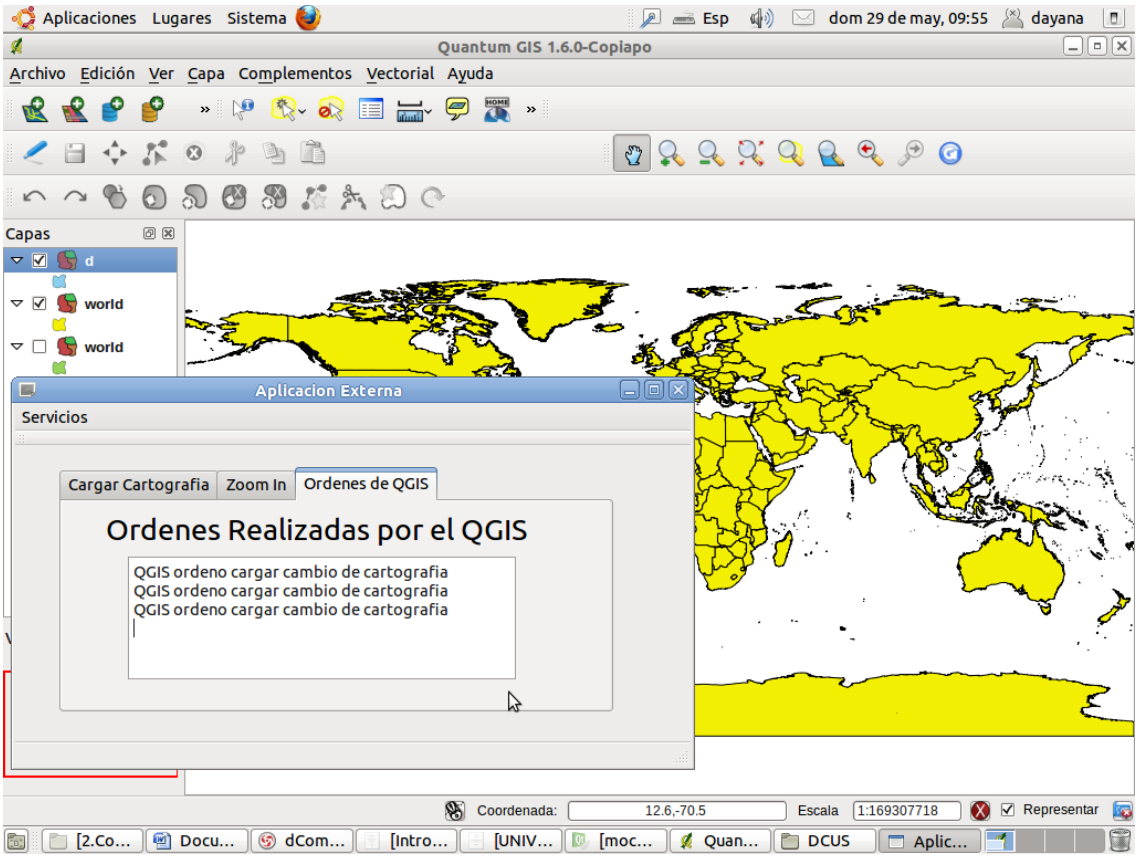
Poscondiciones	Varía el tamaño de la capa.

Tabla 4 Descripción del CU Realizar Solicitud Zoom

Caso de Uso:	**Ordenar Cambio Cartografía**
Actores:	QGIS
Resumen:	El caso de uso inicia cuando el QGIS carga alguna cartografía y ordena al Cliente que se ha cargado una nueva Cartografía.
Precondiciones:	Debe inicializarse la carga de una cartografía.
Referencias	RF 4
Prioridad	Secundario
Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema
	1-El sistema inicializa la carga de una

	cartografía.
2- El Actor envía un mensaje ordenando una nueva cartografía.	3- El sistema transmite mensaje hasta la aplicación externa.

Prototipo de Interfaz



Poscondiciones	Se informa cuando se carga una nueva cartografía.

Tabla 5 Descripción del CU Ordenar Cambio Cartografía

3.6. Conclusiones

En el presente capítulo se describió detalladamente el negocio el cual se modeló a través del Modelo de Dominio y luego se precisaron las características del sistema, los requerimientos funcionales y no funcionales concluyendo que el software a desarrollar es de fácil utilización.

Capítulo 4: Construcción de la Solución Propuesta

El presente capítulo describe la solución en términos de diagrama de clases del diseño para cada caso de uso. Se describe la realización física de los mismos centrándose en los requisitos funcionales y no funcionales. Quedarán definidos los estándares de la interfaz de la aplicación.

4.1. Principios de Diseño

Los principios de diseño en el desarrollo de una aplicación permiten que la misma se convierta en una herramienta útil y atractiva para el usuario. El diseño es algo más que lo que se ve a simple vista, es comunicar una idea con claridad, posibilitando brindar la información adecuada y que le sea útil al usuario.

La aplicación actual cuenta con una pequeña interfaz de usuario, sencilla y legible de manera que exista contraste entre los colores de los textos y el fondo y el tamaño de la fuente sea lo suficientemente adecuado a la vista del usuario. Además se garantiza mensajes de confirmación una vez realizada la introducción de datos al sistema. De manera general se define una apariencia estética de la interfaz, de forma tal que cualquier persona con un mínimo dominio de la computación pueda trabajar con la aplicación sin presentar dificultades notables.

4.2. Arquitectura Lógica del Middleware

En el diseño del Middleware se definieron varios paquetes. Como se había visto anteriormente, con la tecnología middleware que se escogió (ICE), se desarrollará un prototipo para establecer la comunicación entre el QGIS y una Aplicación externa. Para ello se accede a la API del GIS a través de su política extensible mediante plugin, una vez que se tenga este dominio se pueden ejecutar servicios que brinda el Quantum QGIS.

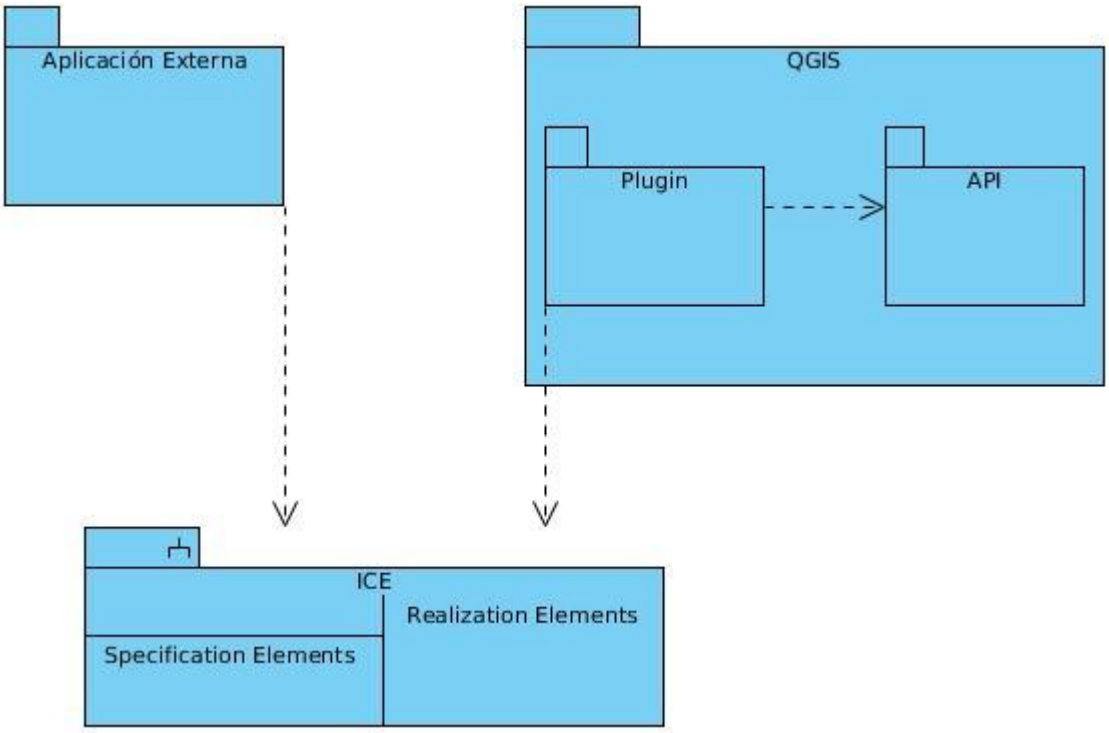


Figura 4 Diagrama de Paquetes del Middleware

4.3. Diagrama de Clases del Diseño

Los diagramas de clases se utilizan para modelar la vista de diseño estática de un sistema, muestran lo que el sistema puede hacer y cómo puede ser construido. A continuación se muestran los diagramas de clases del diseño para cada caso de uso.

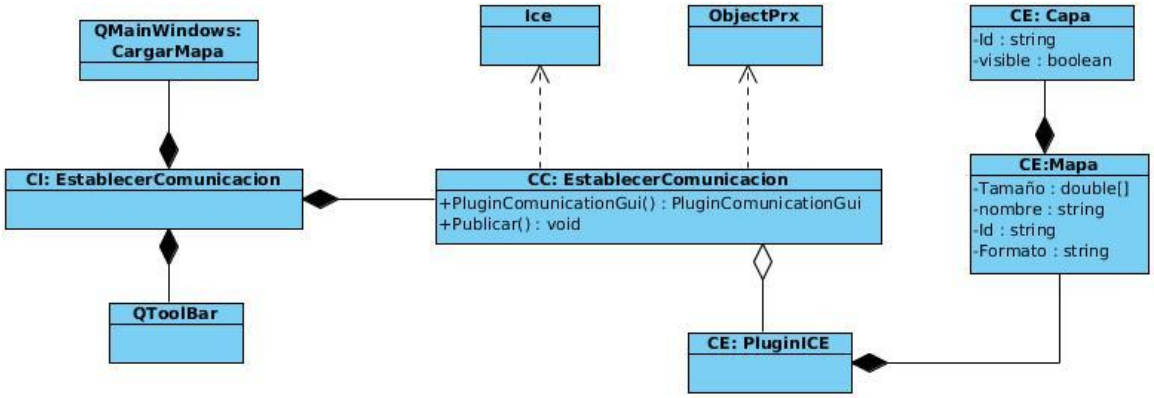


Figura 5 Diagrama de Clases del Diseño Establecer Comunicación

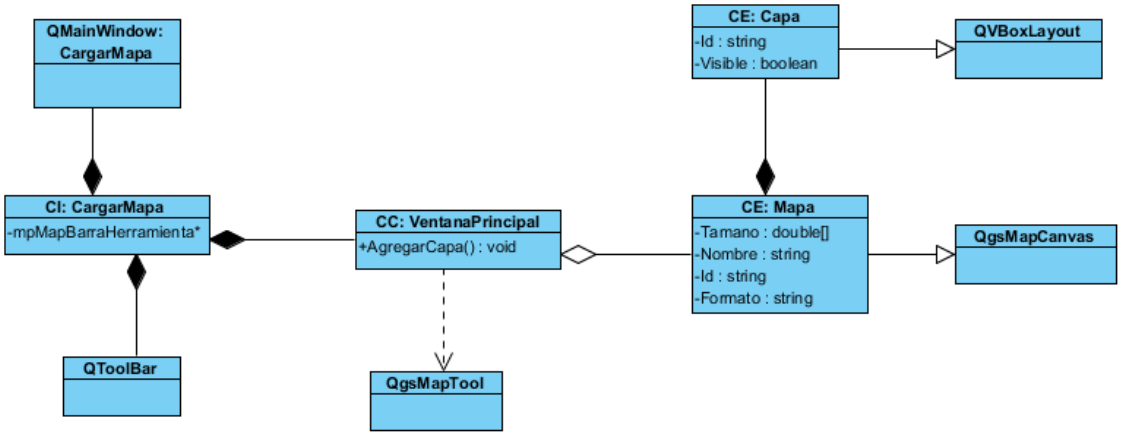


Figura 6 Diagrama de Clases del Diseño del CU Solicitar Servicio Cargar Cartografía

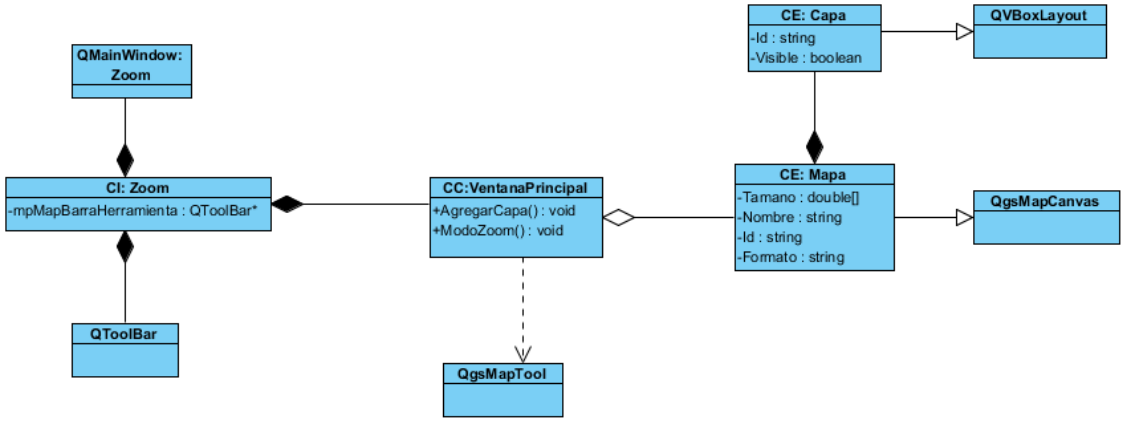


Figura 7 Diagrama de Clases del Diseño del CU Realizar Solicitud Zoom

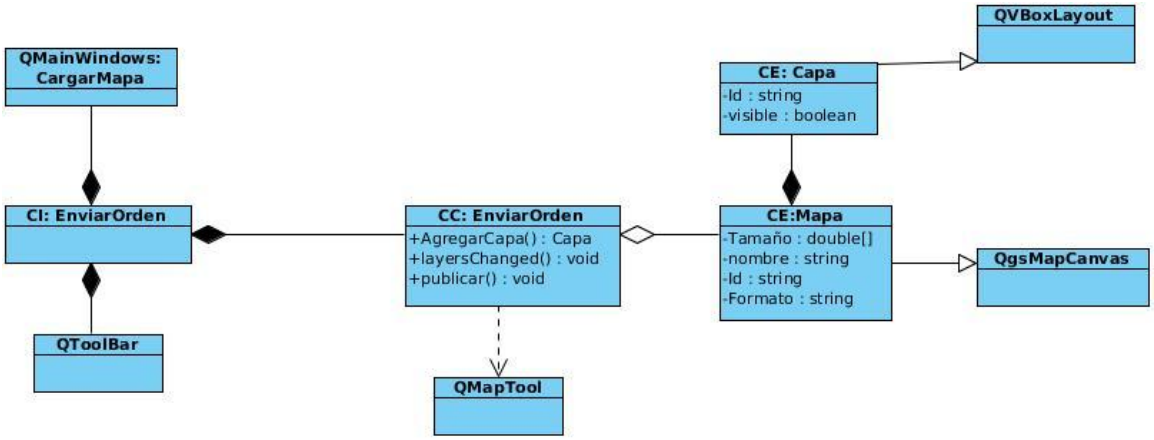


Figura 8 Diagrama de Clases del Diseño del CU Ordenar Cambio Cartografía

4.4. Patrones de Diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Con el paso del tiempo el uso de los patrones se ha ido intensificando ya que gracias a estos se puede producir software más resistente a los cambios. Los patrones de diseño proponen una forma de reutilizar la experiencia de los desarrolladores, usando las prácticas ya empleados por muchos y tomando las mejores propuestas, para así clasifica y describe formas de solucionar problemas que ocurren de forma frecuente en el desarrollo. Por tanto, están basados en la recopilación del conocimiento de los expertos en desarrollo de software. Es una experiencia real, probada y que funciona. Es Historia y ayuda a no cometer los mismos errores [ETECSA, 2009).

Para el desarrollo del sistema se usaron algunos los patrones GRASP (General Responsibility Assignment Software Patterns), que describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades.

Patrón Creador

La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es

encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. El uso de este patrón se evidencia en la clase Mapa, ella es la encargada de crear instancias de la clase Capa.

Patrón Bajo acoplamiento

El término "acoplamiento" hace alusión al grado de dependencia que tienen dos unidades de software y el objetivo del patrón Bajo Acoplamiento es reducir al máximo el acoplamiento entre estas.

Al diseñar el sistema se tuvo como objetivo tener un acoplamiento lo más bajo posible entre las unidades de software lo que permite mantenerlas de una mejor forma, aumenta la reutilización de las mismas, evita el efecto onda, ya que un defecto en una unidad puede propagarse a otras, haciendo incluso más difícil de detectar dónde está el problema y minimiza el riesgo de tener que cambiar múltiples unidades de software cuando se deba alterar una. [Marcello Visconti, 2010)

Patrón Alta cohesión

El término "cohesión" es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. El patrón Alta cohesión dice que la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la clase. Se tuvo como objetivo tener una alta cohesión ya que mejora la claridad y facilidad con que se entiende el diseño, se simplifica el mantenimiento y las mejoras de funcionalidad, a menudo se genera un bajo acoplamiento y soporta mayor capacidad de reutilización.

Patrón Controlador

Un Controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Define además el método de su operación [Marcello Visconti, 2010). En este sistema se reciben eventos de entrada externa, los cuales generalmente incluyen una interfaz gráfica para el usuario (IGU) operado por un cliente. La misma clase controlador debe utilizarse con todos los eventos sistémicos de un caso

de uso, de modo que podamos conservar la información referente al estado del caso. De esta forma se tiene una clase encargada de la lógica del CU. En el desarrollo del sistema se cuenta con una clase controladora para cada CU que es capaz de gestionar la lógica de los datos. (Ver Figura 8)

Patrón proxy

El proxy es un mediador que implica un recubrimiento del objeto real. Es un patrón estructural que tiene como propósito proporcionar un subrogado o intermediario de un objeto para controlar su acceso. Se usa cuando se necesita una referencia a un objeto más flexible o sofisticado que un puntero. Dependiendo de la función que se desea realizar con dicha referencia se puede distinguir diferentes tipos de *proxy* [Marcello Visconti, 2010)*. E*n esta ocasión se emplea el proxy remoto que es el representante local de un objeto remoto, donde permite acceder a un objeto QGIS ocultando el hecho de que un objeto reside en otro espacio de direcciones.

Además, su uso también permite realizar una optimización COW (copy-on-write), puesto que copiar un objeto grande puede ser costoso, y si la copia no se modifica, no es necesario incurrir en dicho gasto. Es útil por tanto para retrasar la replicación de un objeto hasta que cambia.

4.5. Seguridad del Middleware

La seguridad es una consideración importante para muchas aplicaciones distribuidas en redes no confiables como Internet, donde la información es elemento principal a proteger, resguardar y recuperar dentro de las redes empresariales y así garantizar que los recursos informáticos estén disponibles para cumplir sus propósitos. La capacidad de proteger la información confidencial, velar por su integridad, y verificar la identidad de las partes en comunicación es esencial para el desarrollo de aplicaciones seguras (ZeroC, Inc, 2010).

Con esos objetivos, ICE incluye el plugin IceSSL que proporciona estas capacidades mediante el protocolo Secure Sockets Layer (SSL).

El protocolo SSL permite establecer conexiones seguras a través de Internet, de forma sencilla y transparente. La idea consiste en interponer una fase de codificación de los mensajes antes de enviarlos por la red. Una vez que se ha establecido la comunicación, cuando una aplicación quiere enviar información a otra computadora, la capa SSL la recoge y la codifica, para luego enviarla a su destino a través de la red. Con la utilización de IceSSL en el diseño del middleware se asegura un entorno seguro para la comunicación sin sacrificar demasiado el rendimiento y la información enviada a través de la red permanece confiada.

4.6. Generalidades de la Implementación

A partir de los objetivos generales propuestos y de los requisitos expuestos anteriormente, se lleva a cabo la implementación del sistema con el fin de darle cumplimiento a las necesidades.

El modelo de implementación permite planificar las integraciones de sistemas necesarias en cada iteración, distribuye el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue, implementa las clases y subsistemas encontrados durante el diseño y posibilita probar los componentes individualmente para después integrarlos.

4.6.1. Diagrama de despliegue

Despliegue es la etapa del desarrollo que describe la configuración del Sistema para su ejecución en un ambiente del mundo real. Para el despliegue se deben tomar decisiones sobre los parámetros de la configuración, funcionamiento, asignación de recursos, distribución y concurrencia.

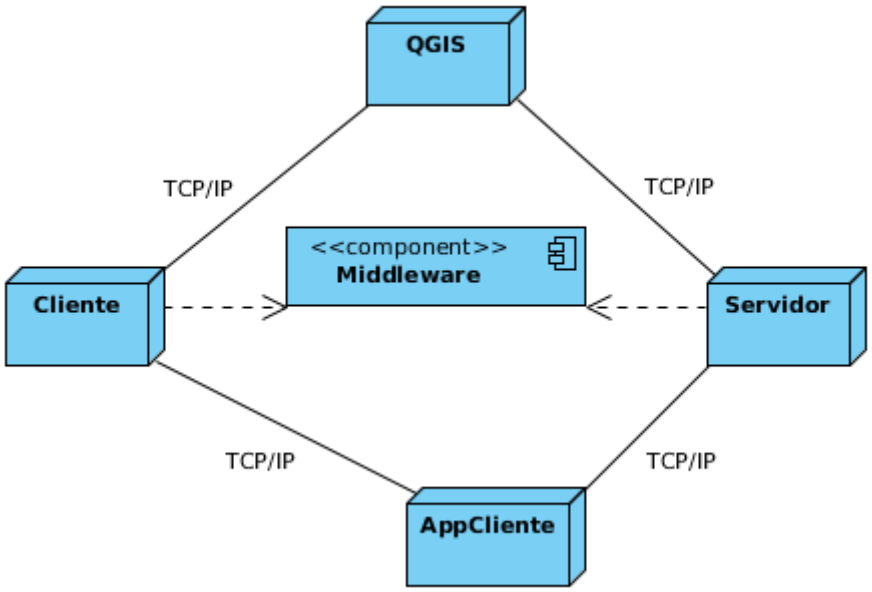


Figura 9: Diagrama de despliegue.

4.6.2. Diagrama de componentes

Los componentes son la parte modular del sistema, desplegable y reemplazable que encapsula implementación y un conjunto de interfaces con su realización. Estos muestran un conjunto de elementos del modelo de implementación tales como los componentes y sus relaciones. Se utiliza para modelar la vista estática del sistema y muestra la organización y las dependencias lógicas entre los componentes de software. Es válido aclarar que por cada clase que se tiene se generaron dos componentes, el .h y el .cpp, esto debido al lenguaje de programación que se utilizó en la implementación del Middleware (C++).

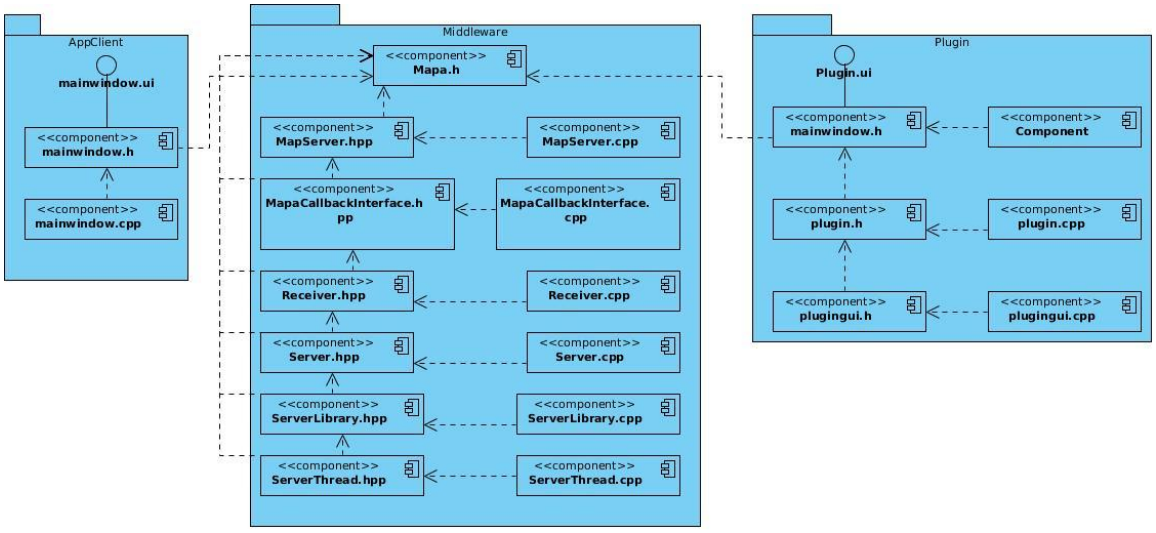


Figura 10 Diagrama de Componentes.

4.7. Estándar de codificación

Las convenciones o estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. A continuación se presentan algunas de estas convenciones para la programación del middleware en lenguaje C++.

Comentarios

Cada programa deberá comenzar con un comentario que incluya:

- Autor.
- Fecha.
- Objetivo o problema que resuelve el programa.
- Fecha de creación.

Cada función debe tener un encabezado que contenga:

- Objetivo de la función y no descripción del procedimiento.
- Comentarios de apoyo a variables, llamadas a función o inclusión de archivos que no sean obvios al proceso.

- Explicación de uso de argumentos (parámetros) no obvios.
- Explicación de uso de valores devueltos (de retorno).

Nombres de identificadores

Se considera como identificador a los nombres de variables (arreglos, matrices, apuntadores), funciones, así como cualquier tipo de dato definido por el usuario (estructura, clase). Dichos identificadores deberán seguir las siguientes normas, además de las definidas por el propio lenguaje.

- Deberán tener un nombre significativo para que por su simple lectura, pueda conocerse su función, sin tener que consultar manuales o hacer demasiados comentarios.
- Para nombres que se usen con frecuencia o para términos largos, se recomienda usar abreviaturas estándar para que éstos tengan una longitud razonable. Si usa abreviaturas deben manejar la misma lógica en todo el programa.
- Evitar identificadores que comiencen con uno o dos caracteres de subrayado para evitar que se confundan con los que el compilador selecciona.

Comenzarán siempre con la primera letra minúscula. Para distinguir palabras dentro del nombre deberá emplearse una letra mayúscula o un guión bajo (_), sin mezclar ambas formas en un mismo programa.

Ejemplos: temperaturaDeVapor o temperatura_de_vapor.

- ❖ Identificadores de punteros (apuntadores)

Su nombre deberá comenzar con la letra p. Ejemplo: pAlumno.

Dónde pAlumno es un puntero que podrá tener la dirección del lugar donde se almacena información de un alumno.

- ❖ Identificadores de variables dimensionadas (arreglos, matrices)

Su nombre deberá comenzar con las letras ar.

Ejemplo: arAlumnos

Donde arAlumnos es un arreglo de datos para guardar información de alumnos.

- Identificadores de funciones

La primera letra deberá ser mayúscula.

Ejemplo: void Funcion( );

Generales

No manejar en los programas más de una instrucción por línea.

- Declarar las variables en líneas separadas.
- Añadir comentarios descriptivos junto a cada declaración de variables, si es necesario.

4.8. Prueba del sistema propuesto

El desarrollo del software implica una serie de actividades en las que la posibilidad de que aparezcan errores humanos es muy común. Los errores pueden comenzar a manifestarse desde el primer momento del proceso en el que los objetivos pueden estar especificados de forma errónea e imperfecta, y así en los posteriores pasos del diseño y desarrollo. Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo del software ha de ir acompañado de una actividad que garantice la calidad. Estas representan una revisión final de las especificaciones del diseño y de la codificación.

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

Pruebas de Caja Negra

Entre los métodos de pruebas que existen, se encuentra el conocido como *Pruebas de Caja Negra.* Este método se refiere a las pruebas que se llevan a cabo sobre la interfaz

del software. Los casos de prueba que se definen pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene. A continuación se presentan los casos de prueba realizados para cada caso de uso, siguiendo el método de caja negra y su técnica Partición de Equivalencia.

Para preparar los casos de pruebas hacen falta un número de datos que ayuden a la ejecución de los estos casos y que permitan que el sistema se ejecute en todas sus variantes, pueden ser datos válidos o inválidos para el sistema según si lo que se desea es hallar un error o probar una funcionalidad. Los datos se escogen atendiendo a las especificaciones del problema, sin importar los detalles internos del sistema, a fin de verificar que el mismo funcione correctamente.

Para desarrollar la prueba de caja negra existen varias técnicas, entre ellas están:

- Técnica de la Partición de Equivalencia: es una técnica que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar [Pressman, 1998).
- Técnica del Análisis de Valores Límites: esta técnica de diseño de casos de prueba completa a la partición equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL (árbol binario de búsqueda) lleva a la elección de casos de prueba en los extremos de la clase. En lugar de centrarse solamente en las condiciones de entrada, el AVL obtiene casos de prueba también para el campo de salida [Presuman, 2002).
- Técnica de Grafos de Causa-Efecto: es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones [Pressman, 2000).

Dentro del método de Caja Negra la técnica de la Partición de Equivalencia es una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas existentes en el sistema, descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico.

Caso de Prueba del CU: **Establecer comunicación**

- *Descripción general del caso de uso:* El caso de uso inicia cuando el usuario necesita intercambiar Información.
- *Condiciones de Ejecución:* Para realizar la comunicación debe existir integración con el QGIS.
- *Secciones a probar en el caso de uso*

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: Comunicar QGIS	EC 1. Solicitar la comunicación.	Esta funcionalidad permite establecer la comunicación con el QGIS, manteniendo el mismo escuchando constantemente las peticiones que se le realizan.	- Iniciar QGIS. - Complementos. - PluginsComunication. - Ok.

Tabla 6: Sección a probar en el Caso de Uso Establecer Comunicación.

- Descripción de Variables: No existen variables para este caso de Pruebas.
- Matriz de datos

ID del escenario	Escenario	Respuesta del Sistema	Resultado de la Prueba
EC 1.	Solicitar la comunicación.	El sistema establece la comunicación del QGIS con una aplicación externa.	Satisfactorio.

Tabla 7: Matriz de datos Comunicar QGIS

Caso de Prueba del CU: **Solicitar servicio Cargar Cartografía**

- *Descripción general del caso de uso:* El caso de uso inicia cuando el cliente selecciona la opción de Cargar una cartografía.
- *Condiciones de Ejecución:* Debe existir una comunicación entre la aplicación externa y el QGIS.
- *Secciones a probar en el caso de uso*

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: Cargar Cartografía	EC 1. Solicitar Cargar Cartografía.	Esta funcionalidad permite desde la aplicación externa cargar una cartografía en el QGIS visualizándose de forma correcta. Este mapa ya está predeterminado.	- Ventana Principal. - Sección Cargar Cartografía. - Botón “Cargar una Cartografía”.

Tabla 8: Sección a probar en el Caso de Uso Solicitar servicio Cargar Cartografía

- Descripción de Variables: No existen variables para este caso de Pruebas.
- Matriz de datos

ID del escenario	Escenario	Respuesta del Sistema	Resultado de la Prueba
EC 1.	Solicitar Cargar Cartografía.	El sistema permite que el QGIS reciba y ejecute la orden de cargar una cartografía.	Satisfactorio.

Tabla 9: Matriz de datos Cargar Cartografía

Caso de Prueba del CU: **Realizar solicitud Zoom**

- *Descripción general del caso de uso:* El caso de uso inicia cuando el cliente selecciona el botón y procede a realizar su función; la posición seleccionada será enviada hacia adelante logrando una mejor visión del elemento seleccionado.
- *Condiciones de Ejecución:* Para lograr una navegación tiene que tener cargada o creada una capa y la misma tiene que estar seleccionada para poder trabajar en ella.
- *Secciones a probar en el caso de uso*

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: Realizar Zoom	EC 1. Solicitar la realización de Zoom.	Esta funcionalidad permite desde la aplicación externa enviar al QGIS una solicitud de realizar un Zoom sobre el mapa seleccionado.	- Ventana Principal. - Sección Zoom In. - Botón “Zoom In”.

Tabla 10: Sección a probar en el Caso de Uso Realizar solicitud Zoom

- Descripción de Variables

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Coordenada X	Caja de Double	NO	Se debe introducir o seleccionar el valor deseado en la coordenada “X”.

2	Coordenada Y	Caja de Double	NO	Se debe introducir o seleccionar el valor deseado en la coordenada “Y”. .

Tabla 11: Descripción de las variables para el caso de uso Realizar solicitud Zoom

- Matriz de datos

ID del escenario	Escenario	Respuesta del Sistema	Resultado de la Prueba
EC 1.	Solicitar la realización de Zoom.	El sistema permite que el QGIS reciba y ejecute la orden de realizar Zoom In sobre la capa que se desee.	Satisfactorio.

Tabla 12: Matriz de datos Realizar Zoom.

Caso de Prueba del CU Dar órdenes de cambios

- *Descripción general del caso de uso:* El caso de uso inicia cuando el QGIS carga alguna cartografía y ordena al Cliente que se ha cargado una nueva Cartografía.
- *Condiciones de Ejecución:* Debe inicializarse la carga de una cartografía.

- *Secciones a probar en el caso de uso*

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: Ordenar Cambio	EC 1. Ordenar cambio de Cartografía.	Esta funcionalidad permite visualizar e informar en la aplicación externa que en el QGIS se ha cargado una nueva cartografía.	- Ventana Principal. - Sección Órdenes de QGIS.

Tabla 13: Sección a probar en el Caso de Uso Ordenar Cambio Cartografía.

- Descripción de Variables: No existen variables para este caso de Pruebas.
- Matriz de datos

ID del escenario	Escenario	Respuesta del Sistema	Resultado de la Prueba
EC 1.	Ordenar cambio de Cartografía.	El sistema informa en la aplicación externa cuando el QGIS carga una nueva Cartografía.	Satisfactorio.

Tabla 14: Matriz de datos Ordenar cambio de Cartografía.

4.9. Conclusiones

En este capítulo se describieron los elementos necesarios para la implementación, a través del diagrama de despliegue que describe cómo se encuentra distribuida física y lógicamente la arquitectura del sistema y sus conexiones. El diagrama de componentes, permitió especificar los componentes establecidos para el desarrollo de la aplicación y la relación entre ellos. Además, se muestran los resultados de las pruebas realizadas al sistema, a partir de los casos de pruebas definidos.

Conclusiones Generales

- Reflejar los conceptos fundamentales para el desarrollo de un sistema ayudan a entender el funcionamiento del entorno de trabajo y el flujo de la información en el mismo.
- Las tecnologías de comunicación estudiadas usan una filosofía muy similar en su funcionamiento, careciendo de una integración a los Sistemas de Información Geográfica.
- El paradigma de Software Libre facilita en gran medida la reutilización de código y un desarrollo acelerado de aplicaciones, posibilitando el estudio exhaustivo de código existente afín con la aplicación que se diseña.
- La comunicación Peer to Peer garantiza un enfoque robusto en el desarrollo de aplicaciones distribuidas, ya que los procesos que se encuentran interconectados asumen un mismo nivel de responsabilidades, dando cumplimiento a todos los requisitos exigidos.
- Para realizar un levantamiento de requisitos cuando no se tienen conocimientos sólidos del negocio abordado, es conveniente valerse de técnicas que permitan validar cada uno de los requisitos, además, cuando se trata de proyectos donde participan un gran número de especialistas es muy conveniente hacer uso de herramientas que faciliten la administración de dichos requisitos.
- Se considera que los Middleware que presentan la arquitectura Object Request Brokers garantizan una escalabilidad, mantenibilidad y flexibilidad al desarrollo de sistemas distribuidos.
- Las pruebas de Caja Negra son un elemento crítico para la garantía de la calidad del software, representando una revisión final de las especificaciones de los requisitos y de la codificación.

Recomendaciones

Con el objetivo de perfeccionar y ampliar las funcionalidades de la herramienta desarrollada se propone al proyecto SIG- Desktop las siguientes recomendaciones:

1. Evaluar la idoneidad del diseño propuesto para su uso en otros Sistemas de Información Geográfica que adolecen de las funcionalidades descritas en el presente trabajo.
2. Capturar y validar nuevos requisitos permitiendo desarrollar nuevas funcionalidades donde se brinde un mayor número de servicios del QGIS, para una mejora continua y desarrollo iterativo e incremental de la propuesta obtenida.
3. Continuar trabajando en el establecimiento de dependencias entre las clases planteadas en el diseño y las clases concretas proporcionadas por el Quantum GIS que faciliten a los desarrolladores soluciones de menor complejidad.
4. Incluir el servicio de cortafuegos para logra eficiencia en redes no seguras.

Bibliografía Referenciada

1. *Protocolo (ciencia de la computación).* [En línea] [Citado el: 10 de enero de 2011.] http://es.encydia.com/pt/Protocolo_(ciencia_de_la_computaci%c3%b3n).
2. Manzano, Liliana. 2009. *¿Sabes qué son las TIC?* sabersinfin.com, pág. http://www.sabersinfin.com/
3. Escuela Técnica Superior de Ingeniería Informática. 2001-2002. Herramientas Web para la enseñanza de protocolos de comunicación. *Herramientas Web para la enseñanza de protocolos de comunicación.* [En línea] 2001-2002. [Citado el: 8 de febrero de 2011.] http://neo.lcc.uma.es/evirtual/cdd/tutorial/Indice.html.
4. Gary E. Sherman, y otros. 2004 - 2007. Quantum GIS. *Quantum GIS.* [En línea] 2004 - 2007. http://www.qgis.org.
5. Gracia, Ing. Joaquin. 2005. Ingeniero Software. *UML: Diagramas UML. ¿Qué es UML? Análisis y Diseño. Ingenieria del Software.* [En línea] 7 de mayo de 2005. [Citado el: 2 de febrero de 2011.] http://www.ingenierosoftware.com/analisisydiseno/uml.php.
6. Humboldt, Alexander von. 2010. Instituto de Investigación de Recursos Biológicos Alexander von Humboldt. *Instituto de Investigación de Recursos Biológicos Alexander von Humboldt.* [En línea] 2010.
7. J., Alonso F. 1998. Curso de metodología de la investigación. Santa Clara : s.n., 1998.
8. Jacobson I, Booch G, Rumbaugh J. 2000. *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison-Wesley, 2000.
9. Leidiana Martínez Diaz, Annier Pérez Ricardo. 2010. *Herramienta para exportar contenidos de Moodle para ser reutilizados en Aplicaciones de Internet Enriquecidas.* s.l. : UCI, 2010.
10. Ltd., Visual Paradigm International. 2007. Free Download Manager. *Visual Paradigm for UML (ME) - (Paradigma Visual para UML (ME)) (Visual Paradigm for UML (ME)) por Visual Paradigm International Ltd. - reporte y descarga.* [En línea] 5 de marzo de 2007. [Citado el: 10 de enero de 2011.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/.
11. Martinto, MSc. Pedro Carlos Pérez. 2009. *El diseño metodológico de la investigación científica.* Ciudad Habana : s.n., 2009.

12. Mora, Sergio Luján. 1998. *C++ paso a paso.* 1998.
13. msdn. 2010. developers code with Microsoft. *developers code with Microsoft.* [En línea] 2010. http://msdn.microsoft.com/es-es/default.aspx.
14. Paradim, Visual. 2010. Visual Paradim Suite. *UML, BPMN and Database Tool for Software Development.* [En línea] 2010. http://www.visual-paradigm.com/.
15. Pleglase, Mario Baños. 2006. *Creación de una herrramienta para comunicación de aplicaciones basadas en UDP(SRC).* Madrid : s.n., 2006.
16. Santiago. 2007. http://geografia.laguia2000.com/general/la-importancia-de-la-geografia-los-sig. *http://geografia.laguia2000.com/general/la-importancia-de-la-geografia-los-sig.* [En línea] La Guía 2000, 27 de marzo de 2007.
17. Sergio Oyarce, Miguel Nussbaum. 2008. *Arquitectura de Sistema de Comunicaciones para Mediaciones Sociales en Redes.* Chile: s.n., 2008.
18. Stroustrup, Bjarne. 1998. *El lenguaje de programación C++.* Madrid: Addison-Wesley, 1998.
19. *ZeroC, Inc. 2010. The Internet Communications Engine (Ice). ZeroC - The Internet Communications Engine (Ice). [En línea] 2010. [Citado el: 26 de 1 de 2011.] http://www.zeroc.com/ice.html.*
20. *Marcello Visconti, Hernán Astudillo. 2010. Fundamentos de Ingeniería de Software. {visconti,hernan} : en inf.utfsm.cl, 2010.*
21. *ETECSA, C. d.-T. (2009). El entorno Protector. Tono , 4.*
22. *Osmany Jorge Riverón. 2007. Modelación de un generador de Informes para Sistemas de Supervición, control y adquisición de datos. s.l. : UCI, 2007.*
23. *Pressman, Rogers. 2002. Ingenería de Software, Parte 2. España : Félix Varela, 2002.*

Bibliografía Consultada

1. *BUSCHMANN Frank, R. M., ROHNERT Hans, SOMMERLAD Peter y STAL Michael. Pattern-Oriented Software Architecture – A System of Patterns. 1996. p.*
2. *CASTILLA, I.C.U.d., Prácticas de la ingeniería de software, Una Herramienta CASE para ADOO. 2007.*
3. *CORNEJO, J. E. G. (2001). Arquitectura en Capas. Un camino hacia los procesos distribuidos. ASSOCIATION, I. S. Recommended practice for architectural description of software-intensive systems 2000. [En línea]. Disponible en: www.standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html*
4. *ÖVERGAARD Gunnar, PALMKVIST Karin. Use Cases Patterns and Blueprints. Addison Wesley Professional, 2004. ISBN 0-13-145134-0.*
5. *RIORDAN, Rebecca (1999). Designing Relational Database Systems. Microsoft Press. ISBN 0-7356-0634-X*
6. *SALAZAR GÓMEZ, Lisset; DUEÑAS NARANJO, Yarely. Sistema automatizado para la captura de información referente al Balance Nacional de Recursos y Reservas de Petróleo de la Oficina Nacional de Recursos Minerales. Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2008.*
7. *STAIR, Ralph M., et al (2003). Principles of Information Systems, Sixth Edition. Thomson Learning, Inc., pp. 132. ISBN 0-619-06489-7*
8. *DATE, C.J. An Introduction to Database Systems. 7th Edition. Reading: Addison-Wesley Publishing Company, 1999.*
9. *DOMÍNGUEZ GENIZ Amalio Javier. Aprenda Reportes con ayuda del plugin IREPORT para Netbeans y MySQL, en N Diapositivas. [En línea]. Recuperado el Disponible en 27 de mayo del 2010. Disponible en: http://ajdgeniz.wordpress.com*
10. *FLANAGAN, David. Java en pocas palabras. Segunda edición. s.l.: O'Relly & Associates. págs. 3-8. ISBN 1-56592-262-X.*
11. *FOWLER Martin, RICE David, FOEMMEL Matthew, HIEATT Edward, MEE Robert, STAFFORD Randy. Patterns of Enterprise Application Architecture. Addison Wesley, ISBN 2002. 0-321-12742-0.*
12. *GAMMA Erich, HELM Richard, JOHNSON Ralph, VLISSIDES John. Elements of Reuseable Object-Oriented Software, New York, 1997.*

Glosario de Términos

1. *Framework:* Es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado.
2. *Middleware*: Es un software de conectividad que hace posible que aplicaciones distribuidas pueden ejecutarse sobre distintas plataformas heterogéneas, es decir, sobre plataformas con distintos sistemas operativos, que usan distintos protocolos de red y, que incluso, involucran distintos lenguajes de programación en la aplicación distribuida. Desde otro punto de vista distinto, un middleware se puede entender como una abstracción en la complejidad y en la heterogeneidad que las redes de comunicaciones imponen. De hecho, uno de los objetivos de un middleware es ofrecer un acuerdo en las interfaces y en los mecanismos de interoperabilidad, como contrapartida de los distintos desacuerdos en hardware, sistemas operativos, protocolos de red, y lenguajes de programación
3. *Multiplataforma*: Término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas.
4. *Plugin*: Es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica.
5. *Sistema*: Un sistema es el conjunto de medios interconectados (objetos, seres humanos, informaciones), utilizados según un proceso dinámico con el fin de alcanzar los objetivos señalados. La IEEE5[3] plantea que: "Sistema es un todo integrado, aunque compuesto de estructuras diversas, interactuantes y especializadas. Cualquier sistema tiene un número de objetivos, y los pesos asignados a cada uno de ellos pueden variar ampliamente de un sistema a otro. Un sistema ejecuta una función imposible de realizar por cualquiera de las partes individuales. La complejidad de la combinación está implícita."
6. *Sistema distribuido*: Un sistema distribuido es una colección de computadoras separadas físicamente y conectadas entre sí por una red de comunicaciones distribuida; cada máquina posee sus componentes de hardware y software que

[3] Standard Dictionary of Electrical and Electronic Terms.

el usuario percibe como un solo sistema (no necesita saber qué hay en las demás máquinas y dónde). Los sistemas distribuidos deben ser muy confiables, ya que si un componente del sistema se descompone otro componente debe de ser capaz de reemplazarlo, esto se denomina Tolerancia a Fallos (ZeroC, Inc, 2010).

7. *Sistema informático:* Un Sistema informático es aquel sistema que se encarga del manejo de la información en cualquier computadora, a través de la cual, el usuario controla las operaciones que realiza el procesador. Se define como la unión que se concibe entre los programas y los componentes físicos, ofreciéndoles respuestas a aquellos usuarios que han hecho de una forma o de otra un determinado pedido (Garcerant, 2008).

Anexo I: Tecnologías de Comunicación entre procesos

Los procesos pueden estar ejecutándose en una o más computadoras conectadas a una red. Las técnicas de IPC (*Inter-process Communication*) están divididas dentro de métodos para: paso de mensajes, sincronización, memoria compartida y llamadas de procedimientos remotos (RPC). El método de IPC usado puede variar dependiendo del ancho de banda y latencia (el tiempo desde el pedido de información y el comienzo del envió de la misma) de la comunicación entre procesos, y del tipo de datos que están siendo comunicados.

- Object Linking and Embedding (OLE)

Es un sistema de objeto distribuido y un protocolo desarrollado por Microsoft que permite a un editor encargar a otro la elaboración de parte de un documento y posteriormente volverlo a importar. Por ejemplo, un sistema de publicación de escritorio puede enviar un poco de texto a un procesador de textos o una imagen a un editor de bitmap usando OLE. La ventaja principal de usar OLE, además de que el tamaño del archivo es menor, es la de poder crear un archivo principal. Se puede hacer una referencia a los datos de ese archivo, con lo que todo cambio posterior en el archivo principal se reflejará en el documento referenciado.

Su uso principal es el manejo de documentos compuestos, pero también puede ser usado para transferir datos entre aplicaciones diferentes usando arrastrar y soltar y operaciones del portapapeles (clipboard). El concepto de "incrustación" ("embedding") es también de uso central en páginas web multimedia, las cuales tienden a contener videos, animaciones y archivos de música dentro del código HTML. Sin embargo, OLE usa una arquitectura denominada fat client (cliente pesado), la cual significa que el tipo de archivo o la aplicación que va a ser incrustada debe estar presente en la máquina en la cual esta va a trabajar. Por ejemplo, si una hoja de cálculo de Microsoft Excel está por ser procesada o incluso solo visualizada, debería haber una copia de Excel o un visor de Excel instalado en la máquina del usuario.

- Dynamic Data Exchange (DDE).

El Intercambio Dinámico de datos es un mecanismo soportado por Windows que permite a dos aplicaciones mantener una conversación durante la cual, la información

es intercambiada automáticamente. DDE automatiza el cortar y pegar información entre aplicaciones Windows. La forma en la que trabaja DDE consiste en que una aplicación Windows llamada *destino* (cliente en versiones anteriores), que es la aplicación que inicia la conversación, le dice a otra aplicación Windows, llamada *fuente* (servidor en versiones anteriores), que quiere información, es decir, que quiere establecer una conversación DDE o un enlace DDE. Una aplicación puede mantener varias conversaciones DDE al mismo tiempo. En Visual Basic los formularios pueden ser fuentes DDE o destinos DDE cambiando su propiedad *LinkMode* a *vbLinkNone* o *vbLinkSource*. Las cajas de texto, las de figuras (picture box) y las etiquetas, pueden ser destinos DDE. Normalmente, la información fluye de la fuente al destino. Es necesario conocer el nombre de la aplicación origen a la que se va a llamar. A continuación, se necesita conocer el tema de la conversación DDE, que suele ser una unidad de información que reconoce la fuente. Con la aplicación y el tema, se identifica una conversación. Y, por último, queda establecer el elemento de conversación DDE o pieza de información que se va a pasar en la conversación, significativa para ambas aplicaciones. Sin embargo DDE tiene limitaciones en performance y confiabilidad para tratar con información en tiempo real.

- RPC (Remote Procedure Call / llamada a un procedimiento remoto).

Permite que los programas realicen llamadas a funciones localizadas en otras máquinas. Los programadores no se tienen que preocupar por los detalles de la programación de la red.

Desde el punto de vista de un programador la llamada a una función remota es y funciona de la misma manera que lo haría si la llamada fuese local. En este sentido, se logra transparencia.

Cada función pasa a tener dos partes: cliente, la máquina local donde se implementa la interface (prototipo de una función) para invocar las funciones remotas. Servidor, implementación de las funciones propiamente dichas.

RPC tiene gran apoyo para la conversión automática de datos y de las comunicaciones con otros sistemas operativos. A través de RPC, puede crear de alto rendimiento, estrechamente unida aplicaciones distribuidas.

Las comunicaciones RPC tienen como *desventaja* que se basan en la idea que el receptor está operativo para poder invocar una cierta función, no podemos suponer que el receptor siempre estará operativo y esperando a comunicarse. Esta comunicación la hace a través de eventos enviados por la red constantemente, si el paquete es muy grande y en una red local, puede traer retrasos en la misma. La solución es definir la comunicación en término de paso de mensajes.

- En Java, se utiliza un mecanismo conocido como RMI (Remote Method Invocation)

Es un mecanismo de expansión de RPC cuyo objetivo es dar soporte a sistemas orientado a objetos. La idea es tener objetos distribuidos. Para acceder al estado de un objeto sólo se realiza a través de métodos definidos por un objeto interface. Un objeto ofrece múltiples interfaces. Una interface puede ser implementada por múltiples objetos.

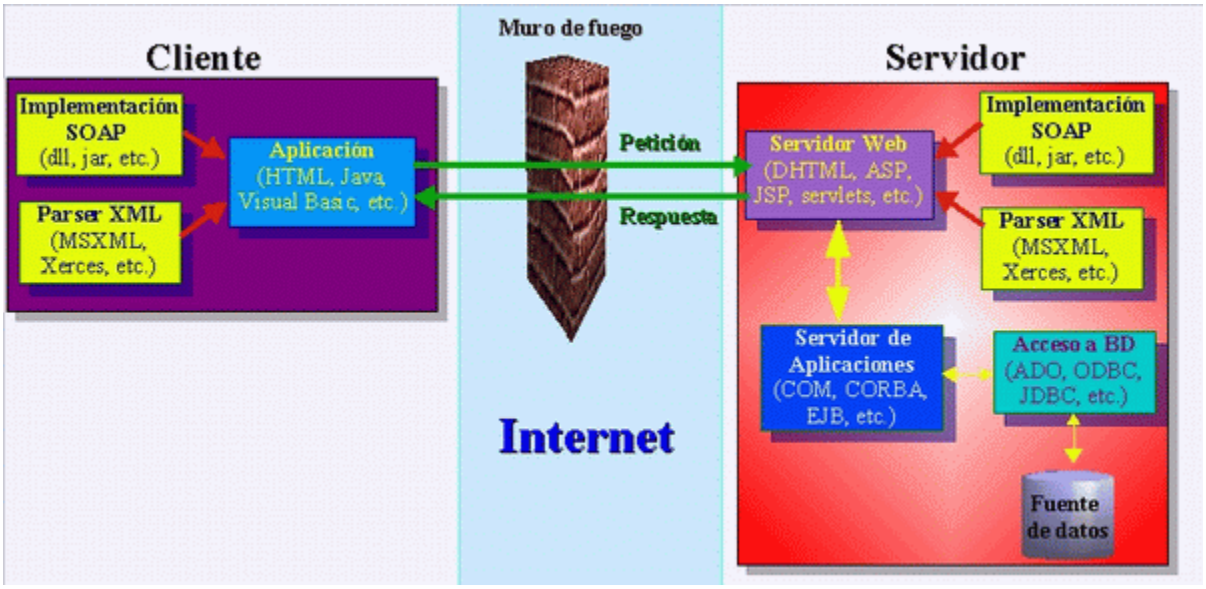
- En la plataforma .NET, se utiliza .NET Remoting(un mecanismo similar a RMI)

Permite crear fácilmente aplicaciones ampliamente distribuidas. Se puede construir aplicaciones cliente que utilizan los objetos en otros procesos en el mismo equipo o en cualquier otro equipo que es accesible a través de su red. También puede utilizar. NET Remoting para comunicarse con otros dominios de aplicación en el mismo proceso y tiene gran capacidad para poder trabajar desde una máquina con los objetos en memoria de la máquina Remota.

Es flexible y fácilmente personalizable. Se puede reemplazar un protocolo de comunicación con otro, o un formato de serialización con otro sin tener que recompilar el cliente o el servidor. Puede comunicarse desde una aplicación web, una aplicación de consola, un servicio de Windows de casi cualquier cosa que desee utilizar. Cualquier aplicación puede alojar objetos remotos y ofrecer sus servicios a cualquier cliente en su computadora o red.

- *Simple Object Access Protocol* (SOAP)

SOAP (Protocolo Simple de Acceso a Objetos) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML, entre diversas aplicaciones Web desarrolladas bajo tecnologías diferentes, tomando como base protocolos ya establecidos y con gran aceptación en Internet, como HTML(para la transmisión) y XML(para la codificación de datos). Se adhiere al paradigma de la programación orientada a objetos. Además SOAP especifica el formato de mensaje que accede e invoca a los objetos.



Funcionamiento de SOAP

La especificación SOAP menciona que las aplicaciones deben ser independientes del lenguaje de desarrollo, por lo que las aplicaciones cliente y servidor pueden estar escritas con HTML, DHTML, Java, Visual Basic u otras herramientas y lenguajes disponibles.

Entre las ventajas que tiene el SOAP podemos encontrar que

- Es sencillo de implementar, probar y usar.
- Atraviesa "firewalls" y routers, pues estos "piensan" que es una comunicación HTTP.

- Tanto los datos como las funciones se describen en XML, lo que permite que el protocolo no sólo sea más fácil de utilizar sino que también sea muy sólido.
- Es independiente del sistema operativo y procesador.
- Se puede utilizar tanto de forma anónima como con autenticación (nombre/clave).
- Facilidad para utilizar cualquier lenguaje: Los desarrolladores involucrados en nuevos proyectos pueden elegir desarrollar con el último y mejor lenguaje de programación que exista. SOAP no especifica una API, por lo que la implementación de la API se deja al lenguaje de programación, como en Java, y la plataforma como Microsoft .Net.
- No se encuentra fuertemente asociado a ningún protocolo de transporte: La especificación de SOAP no describe como se deberían asociar los mensajes de SOAP con HTTP. Un mensaje de SOAP no es más que un documento XML, por lo que puede transportarse utilizando cualquier protocolo capaz de transmitir texto.
- No está atado a ninguna infraestructura de objeto distribuido: La mayoría de los sistemas de objetos distribuidos se pueden extender, y alguno de ellos admiten SOAP.
- Aprovecha los estándares existentes en la industria: Los principales contribuyentes a la especificación SOAP evitaron, intencionalmente, reinventar las cosas. Optaron por extender los estándares existentes para que coincidieran con sus necesidades. Por ejemplo, SOAP aprovecha XML para la codificación de los mensajes, en lugar de utilizar su propio sistema de tipo que ya están definidas en la especificación esquema de XML. Y como ya se ha mencionado SOAP no define un medio de trasporte de los mensajes, los mensajes de SOAP se pueden asociar a los protocolos de transporte existentes como HTTP y SMTP.
- Permite la interoperabilidad entre múltiples entornos: SOAP se desarrolló sobre los estándares existentes de la industria, por lo que las aplicaciones que se ejecuten en plataformas con dichos estándares pueden comunicarse mediante mensaje SOAP con aplicaciones que se ejecuten en otras plataformas. Por ejemplo, una aplicación de escritorio que se ejecute en un PC puede comunicarse con una aplicación del *back-end* ejecutándose en un mainframe capaz de enviar y recibir XML sobre HTTP.

Entre las desventajas de SOAP se tiene que:

- Las desventajas de la utilización de SOAP recaen en la dificultad para entender las especificaciones del protocolo, puesto que es un complejo esquema de codificación en el cual es necesario precisar que todos los mensajes se incluyan en un sobre, con el contenido del mensaje dentro de un elemento de cuerpo para que puedan ser entendidos por cada una de las aplicaciones Web que procesan el mensaje.
- SOAP convierte en opcionales elementos como encabezados y ofrece un amplio margen con respecto a lo que se puede incluir en el elemento de cuerpo y además cambia los nombres de métodos en etiquetas secundarias del cuerpo y los argumentos en etiquetas secundarias del nombre del método, lo que puede generar ciertos problemas de interoperabilidad.
- Las especificaciones SOAP indican que si recibe un encabezado SOAP con un atributo mustUnderstand establecido como "1", deberá entenderlo o generar un error. Numerosas implementaciones no lo hicieron al principio lo que implicó problemas de interoperabilidad.

- Common Object Request Broker (CORBA)

Desde sus principios, el objetivo de CORBA fue permitir la interconexión abierta de distintos lenguajes, implementaciones y plataformas. De esta forma, CORBA cumple con las cuatro propiedades enumeradas como deseables de los middleware. Para lograr estos objetivos, la OMG decidió no establecer estándares binarios, todo está estandarizado para permitir implementaciones diferentes y permitir que aquellos proveedores que desarrollan CORBA puedan ofrecer valor agregado. La contrapartida es la imposibilidad de interactuar de manera eficiente a nivel binario. Todo producto que sea compatible con CORBA debe utilizar los costosos protocolos de alto nivel.

CORBA está constituido esencialmente de tres partes: un conjunto de interfaces de invocación, el ORB y un conjunto de adaptadores de objetos. CORBA va más allá de simples servicios middleware, provee una infraestructura para construir aplicaciones orientadas a objetos. Las interfaces definen los servicios que prestan los objetos, el ORB se encarga de la localización e invocación de los métodos sobre los objetos y el adaptador de objeto es quien liga la implementación del objeto con el ORB.

Para que las interfaces de invocación y los adaptadores de objetos funcionen correctamente, se deben cumplir dos requisitos importantes. En primer lugar, las interfaces de los objetos deben describirse en un lenguaje común. En segundo lugar, todos los lenguajes en los que se quieran implementar los objetos deben proveer un mapeo entre los elementos propios del lenguaje de programación y el lenguaje común. La primera condición permite generalizar los mecanismos de pasaje de parámetros (marshaling y unmarshaling). La segunda permite relacionar llamadas de o a un lenguaje en particular con el lenguaje de especificación común. Este lenguaje común fue una parte esencial de CORBA desde sus orígenes y es conocido como el OMG IDL: Interface Definition Language.

CORBA automatiza muchas tareas comunes y "pesadas" de programación de redes tales como registro, localización y activación de objetos, manejo de errores y excepciones, codificación y decodificación de parámetros, y protocolos de transmisión.

Ventajas de CORBA:

- Interfaz independiente del lenguaje de programación.
- Integración de herencia.
- Infraestructura de objetos distribuidos.
- Transparencia en la localización.
- Transparencia en la red.
- Comunicación directamente con los objetos.
- Interfaz de invocación dinámica.

- Internet Communication Engine (ICE)

ICE es un estándar desarrollado por ZeroC para el desarrollo de aplicaciones basada en objetos distribuidos, surge como alternativa del Common Object Request Broker y difiere de él en algunos conceptos claves. ICE es un middleware ligero, abierto y orientado a objetos, es decir, ICE proporciona herramientas, APIs y soporte de bibliotecas para construir aplicaciones cliente-servidor orientadas a objetos. Una aplicación ICE se puede usar en entornos heterogéneos: los clientes y los servidores pueden escribirse en diferentes lenguajes de programación (C, C++, Java, C#, Visual Basic, Phyton, PHP), pueden ejecutarse en distintos sistemas operativos (Windows,

Linux, y otras propietarias) y en distintas arquitecturas, y pueden comunicarse empleando diferentes tecnologías de red. Además, el código fuente de estas aplicaciones puede portarse de manera independiente al entorno de desarrollo.

Por defecto, el modelo de envío de peticiones utilizado por ICE está basado en la llamada síncrona a procedimientos remotos: una invocación de operación se comporta como una llamada local a un procedimiento, es decir, el hilo del cliente se suspende mientras dure la llamada y se vuelve activar cuando la llamada se ha completado (y todos los resultados están disponibles).

ICE también proporciona invocación de métodos asíncrona (asynchronous method invocation) o AMI: un cliente puede invocar operaciones de manera asíncrona, es decir, el cliente utiliza un proxy de la manera habitual para invocar una operación pero, además de pasar los parámetros necesarios, también pasa un objeto de retro llamada (callback object) y retorna de la invocación inmediatamente. Una vez que la operación se ha completado, el núcleo de ejecución de la parte del cliente invoca a un método en el objeto de retro llamada pasado inicialmente, proporcionando los resultados de la operación a dicho objeto (o en caso de error, proporciona la información sobre la excepción asociada).

El servidor no es capaz de diferenciar una invocación asíncrona de una síncrona. De hecho, en ambos casos el servidor simplemente aprecia que un cliente ha invocado una operación en un objeto.

El tratamiento de métodos asíncrono es útil, por ejemplo, si un servidor ofrece operaciones que bloquean a los clientes por un periodo largo de tiempo. Por ejemplo, el servidor puede tener un objeto con una operación get que devuelva los datos de una fuente de datos externa y asíncrona, lo cual bloquearía al cliente hasta que los datos estuvieran disponibles.

El adaptador de objetos es una parte de la API de ICE específico al lado del servidor: sólo los servidores utilizan los adaptadores de objetos. Un adaptador de objetos tiene varias funciones, como traducir las peticiones de los clientes a los métodos específicos al lenguaje de programación empleado, asociarse a uno o más puntos finales de transporte, o crear los proxies que pueden pasarse a los clientes.

Las interfaces, las operaciones, y los tipos de datos intercambiados entre el cliente y el servidor se definen utilizando el lenguaje Slice (Specification Languagefor ICE). Slice permite definir el contrato entre el cliente y el servidor de forma independiente del lenguaje de programación empleado. Las definiciones Slice se compilan por un compilador a una API para un lenguaje de programación específico, es decir, la parte de la API que es específica a las interfaces y los tipos que previamente han sido definidos y que consisten en código generado.

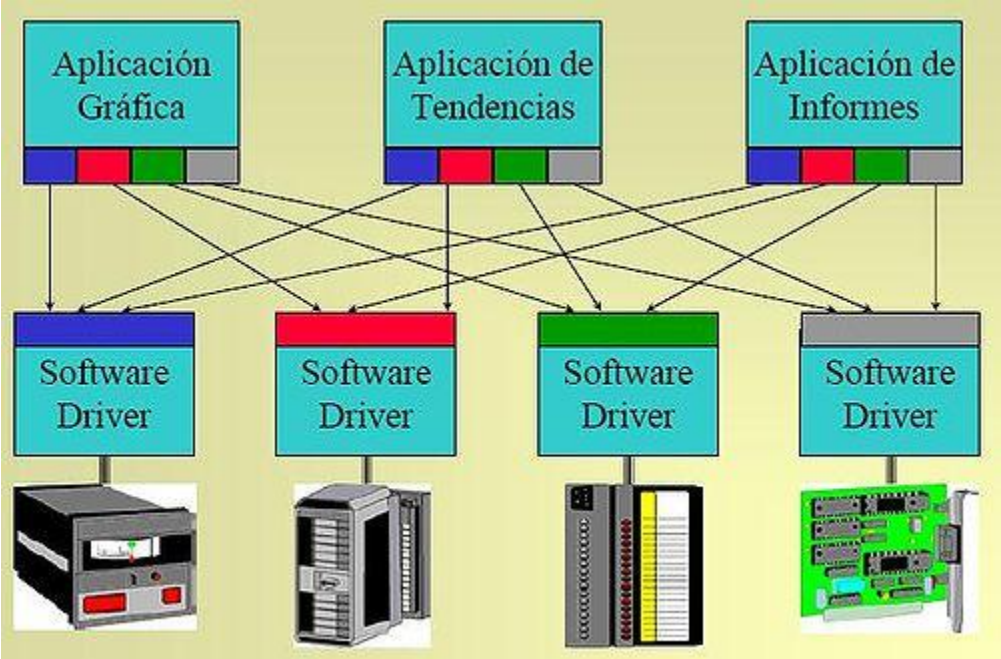
El núcleo de ICE proporciona una sofisticada plataforma cliente-servidor para el desarrollo de aplicaciones distribuidas. También maneja una serie de servicios que gestionan la activación de servidores bajo demanda, la distribución de proxies a los clientes, la distribución de eventos asíncronos, la configuración de aplicaciones, etc. Manteniendo estos servicios disponibles como parte de la plataforma permite al desarrollador centrarse en el desarrollo de la aplicación en lugar de construir la infraestructura necesaria en primer lugar. Algunos de estos servicios son [ZeroC, 1998]:

- *IceGrid:* es la implementación de un servicio de localización ICE.
- *IceBox*: es un único servidor de aplicaciones que permite gestionar el arranque y la parada de un determinado número de componentes de aplicación.
- *IceStorm:* es un servicio de publicación-subscripción que actúa como un distribuidor de eventos entre servidores y clientes. Los publicadores envían eventos al servicio IceBox, el cual notifica a los subscriptores. De esta forma, un único evento publicado por el publicador se puede enviar a múltiples subscriptores. Los eventos están categorizados en función de un tema, y los subscriptores especifican los temas en los que están interesados. IceBox permite seleccionar entre distintos criterios de calidad de servicio para que las aplicaciones puedan obtener una relación fiabilidad-rendimiento apropiada.
- *IcePatch2:* es un servicio que permite la fácil distribución de actualizaciones de software a los clientes.
- *Glacier2:* es el servicio de cortafuegos de ICE, el cual permite que tanto los clientes como los servidores se comuniquen de forma segura a través de un cortafuegos sin comprometer la seguridad. El tráfico entre el cliente y el servidor queda completamente encriptado utilizando certificados de clave pública y es bidireccional. Glacier2 también proporciona soporte para la autenticación mutua y para la gestión segura de sesiones.

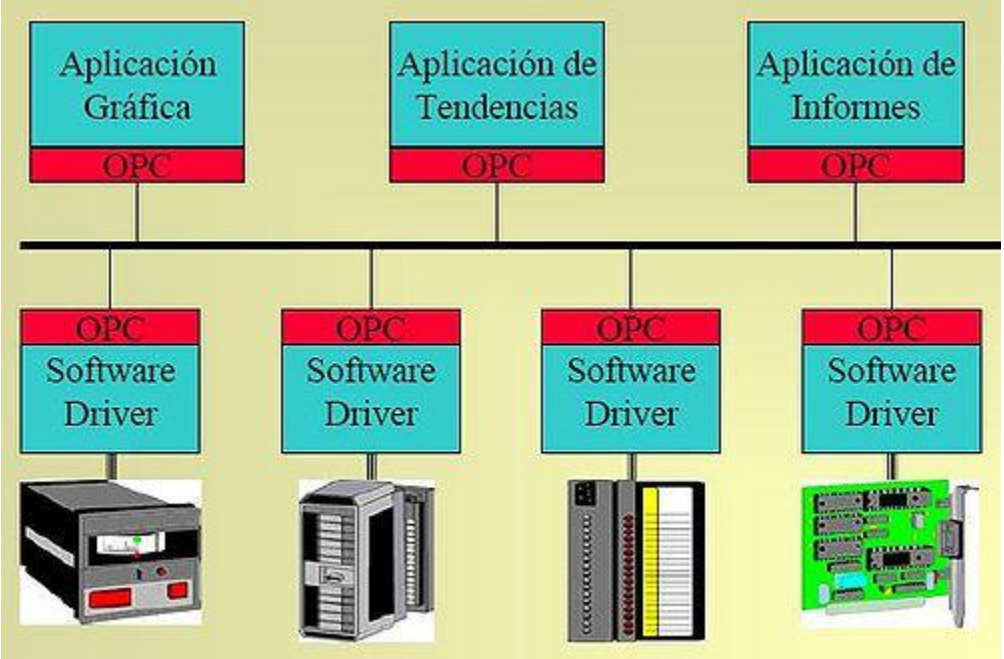
- *IceSSL:* Un SSL dinámico que transporta extensiones para el núcleo de ICE. Ofrece autenticación, cifrado e integridad en los mensajes utilizando el protocolo estándar de la industria SSL.

- OLE for Process Control (OPC)

El OPC 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 servidor 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.



El problema sin tecnología OPC.



La solución al problema al contar con tecnología OPC.

La arquitectura OPC es un modelo Cliente-Servidor donde el Servidor OPC proporciona una interfaz al objeto OPC y lo controla. Una aplicación cliente OPC se comunica a un servidor OPC a través de un cliente OPC específico por medio de una interfaz de automatización. El servidor OPC lleva a cabo la interfaz cliente, y opcionalmente lleva a cabo la interfaz de automatización.

Una aplicación cliente OPC, puede conectarse por medio de una red, a varios servidores OPC proporcionados por uno o más fabricantes. De esta forma no existe restricción por cuanto a tener un Software Cliente para un Software Servidor, lo que es un problema de interoperabilidad que hoy en día se aprecia con sistemas del tipo propietario.

La comunicación OPC está basada en diversos componentes de los sistemas Operativos Microsoft. Estos componentes funcionan perfectamente dentro de una típica red LAN en condiciones estándar (buen ancho de banda y conexiones constantes), pero bajo condiciones poco favorables el comportamiento de las comunicaciones puede llevar a interrupciones en el envío o a una pérdida de datos.

Para este inconveniente utilizamos OPC Tunneller que provee una alternativa que elimina los riesgos y problemas originados por el DCOM. Esta tecnología utiliza comunicaciones Standard TCP/IP en lugar de DCOM para transportar mensajes con datos OPC. De esta forma obtenemos algunas ventajas tales como: se pueden enviar y recibir datos a través de diferentes dominios; permite una mejor seguridad de los Firewalls y reduce considerablemente los requerimientos de Ancho de Banda.

- Socket

Un socket (enchufe), (Sistema fuertemente acoplado a las redes TCP/IP) es un método para la comunicación entre un programa del cliente y un programa del servidor en una red. Es un fichero existente en la máquina cliente y en la máquina servidora, en el lado del cliente se utiliza la clase Socket y en el del servidor el Server Socket, que sirve en última instancia para que el programa servidor y el cliente lean y escriban la información. Esta información será la transmitida por las diferentes capas de red.

Los sockets se crean y se utilizan con un sistema de peticiones o de llamadas de función a veces llamados interfaz de programación de aplicación de sockets, para los cuales se necesita un protocolo de comunicaciones, que permite el intercambio de octetos, una dirección del Protocolo de Red (Dirección IP, si se utiliza el Protocolo TCP/IP, aunque también es posible utilizar UDP o IPX), que identifica una computadora y un número de puerto, que identifica a un programa dentro de una computadora.

En una red de ordenadores hay varios ordenadores que están conectados entre sí por un cable. A través de dicho cable pueden transmitirse información. Es claro que deben estar de acuerdo en cómo transmitir esa información, por eso se usa generalmente el TCP/IP. Al establecer la comunicación entre los programas si uno de ellos está atareado en otra cosa y no atiende la comunicación, el otro quedará bloqueado hasta que el primero lea o escriba los datos.