Universidad de las Ciencias Informáticas

Facultad 2

Título: Propuesta de Front End para gestión de posicionamiento vehicular sobre redes TETRA NEBULA.

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autor: Yosel Hernández Escalona

Tutores: Ing. Alberto Arce Martínez

Ing. Leosdany Sánchez Alba

Cuidad de La Habana, Junio del 2011

Donde hay soberbia, allí habrá ignorancia; mas donde hay humildad, habrá sabiduría.

Rey Salomón.

DECLARACIÓN DE AUTORÍA.

Declaro que soy el único autor de este trabajo y autorizo al Centro de Telemática de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____ _____

Yosel Hernández Escalona Ing. Alberto Arce Martínez

_____

Ing. Leosdany Sánchez Alba

DATOS DE CONTACTO.

Leosdany Sánchez Alba.

Ingeniero en Ciencias Informáticas.

Categoría Docente: Recién Graduado en Adiestramiento.

Años de experiencia en el tema: 2.

Alberto Arce Martínez

Ingeniero en Ciencias Informáticas.

Categoría Docente: Recién Graduado en Adiestramiento.

Años de experiencia en el tema: 2.

AGRADECIMIENTOS.

A Dios, por permitirme realizar cosas increíbles que no estaban a mi alcance.

A todas las personas que se han preocupado por mí y han contribuido en mi formación profesional y personal.

DEDICATORIA.

A las personas que más admiro, mis padres.

RESUMEN.

Los sistemas AVL -*Automatic Vehicle Location / Localización Automática de Vehículos*- son de los servicios más demandados por las empresas en todo el mundo. Han alcanzado un gran nivel de penetración en los sectores públicos y de emergencias con la gran variedad de servicios que brindan gracias a los estándares digitales que hoy se implementan. La empresa Teltronic posee la Serie CE-CO-CO que gestiona la información que se tiene en el receptor GPS de un dispositivo y la transmite a un centro de control donde se encuentra instalado un SIG -*Sistema de Información Geográfica / Geographic Information System* - para monitorizar los recursos.

La Serie CE-CO-CO no puede integrarse con otros SIG. Debido a esta problemática la Universidad de las Ciencias Informáticas, específicamente el Centro de Telemática, propone la realización de un Front End de comunicaciones que permita el intercambio de información procedente del servidor AVL NEBULA con los SIG. Este sistema posibilitará la utilización de otros SIG para la monitorización de los recursos.

En este documento se especifican los resultados del diseño y la implementación del sistema "FE AVL - Front End AVL-". Para ello se utilizaron herramientas que facilitan el desarrollo de la aplicación siguiendo el principio de software libre.

Palabras Clave: *GIS, GPS, NEBULA, Sistemas AVL.*

ÍNDICE.

INTRODUCCIÓN

La Universidad de las Ciencias Informáticas (UCI) es un proyecto cuya misión fundamental es formar profesionales calificados en la rama de la Informática, está conformada por centros de desarrollos con perfiles dedicados a los sectores más importantes de la economía del país y del mundo.

En el centro de Telemática se desarrollan proyectos nacionales e internacionales con diferentes empresas entre las que se encuentra la empresa española de Telecomunicaciones TELTRONIC la cual estableció relaciones con la universidad desde el mes de Julio del 2008. Esta empresa utiliza el estándar europeo TETRA -*Terrestrial Trunked Radio / Radio Troncal Terrestre*- que ofrece facilidades para la transmisión de datos, haciendo posible el desarrollo de aplicaciones de telemetría, acceso a bases de datos y sistemas AVL. Los sistemas AVL son unos de los servicios más demandados en redes de radio profesional, debido a que gestiona la información que se tiene en el receptor GPS -*Global Position System / Sistema de Posicionamiento Global*- de un dispositivo y la transmite a un centro de control. En el centro de control se encuentra un ordenador que posee un SIG que se encarga de representar sobre un mapa la información geográfica recibida en tiempo real, agilizando los procesos de búsqueda y control pues es capaz de realizar cálculos de distancia, ruta mínima entre otras funcionalidades.

En la actualidad la empresa TELTRONIC cuenta con una solución que le permite gestionar las posiciones de los dispositivos móviles usando el servidor AVL NEBULA. Este servidor le proporciona la información a un SIG que la representa en tiempo real. Debido a que esta solución no tiene la capacidad de integrarse con otros SIG surge como **problema científico** la siguiente interrogante: ¿Cómo lograr el intercambio de información procedente del servidor AVL NEBULA con diferentes SIG?

Con este trabajo se pretende diseñar un sistema que permita la comunicación del servidor AVL NEBULA con diferentes SIG, dándole la posibilidad al usuario de elegir el de su preferencia. El sistema será capaz de exportar la información a formato KML -*Keyhole Markup Language/ Lenguaje de Marcado de Keyhole* - y además compartirla a través del puerto socket con el servicio TIG-*Trama de Información Geográfica*-.

Por tanto el **Objeto de Estudio** de este trabajo está relacionado con el proceso de comunicación de los servidores AVL y su interacción con los SIG.

El **Campo de Acción** queda enmarcado específicamente en el proceso de comunicación del servidor AVL NEBULA y su interacción con los SIG.

Se persigue como **Objetivo General** implementar el proceso de comunicación con el servidor AVL NEBULA y la decodificación de la información proveniente de este para la creación de ficheros KML y servicios TIG.

Para cumplir el objetivo trazado, se desarrollaron las siguientes tareas:

- Estudio del protocolo de comunicación socket.
- Estudio de los comandos del servidor AVL NEBULA.
- Estudio de las soluciones actuales de TELTRONIC para el posicionamiento vehicular.
- Estudio de la estructura de los formatos para la representación de la información geográfica.
- Realización de un formato de trama para la comunicación por socket con SIG externos.

Para lograr el intercambio de información procedente del servidor AVL NEBULA con diferentes SIG, se aplicaron los siguientes **métodos de investigación:**

Métodos Teóricos:

Analítico – sintético: Posibilita procesar la información para guiar la investigación sobre los servidores AVL y los SIG, simplificando y organizando el análisis de todos los datos recopilados.

Histórico – lógico: Permite realizar un estudio de los antecedentes y tendencias actuales de los servidores AVL y los SIG.

Métodos Empíricos:

Experimento: Permite realizar pruebas al sistema, para verificar el correcto funcionamiento de las funcionalidades implementadas.

Con el propósito de organizar y darle una estructura al trabajo se ha decidido dividirlo en 4 capítulos.

Capítulo 1: Se expone un estudio sobre los servidores AVL y los SIG. Además se detallan las herramientas y tecnologías que se utilizarán en el desarrollo del sistema, así como los lenguajes y metodología de desarrollo utilizados.

Capítulo 2: Describe el problema, la situación problémica y los procesos que serán objeto de automatización. Además aborda aspectos esenciales del dominio y los requerimientos a tener en cuenta.

Capítulo 3: Se realizan los diagramas de clases del diseño y diagramas de paquetes del diseño. Además se definen los patrones a utilizar y las descripciones de las principales clases.

Capítulo 4: Se confecciona el diagrama de despliegue, el diagrama de componentes y se describen las pruebas realizadas al sistema.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

Introducción.

En el presente capítulo se describe el estado del arte de las soluciones que gestionan la comunicación del servidor AVL NEBULA con los SIG que se han estudiado. Además se incluyen conceptos teóricos de las tecnologías que se utilizarán para implementar la solución que se propone en esta investigación. Se abordarán los temas relacionados con la interfaz de comunicación que permiten intercambiar información entre una aplicación externa y el servidor AVL NEBULA, así como los formatos a los que se exportará la información. También se tratará la metodología de desarrollo de software que fue utilizada para implementar el ciclo de desarrollo de la solución.

1.1 Infraestructura de los sistemas de posicionamiento vehicular.

La infraestructura de los sistemas de posicionamiento vehicular está compuesta por tres sistemas, el servidor AVL, el Front End de comunicación y los SIG. En estos sistemas tanto los SIG como los servidores AVL realizan su trabajo de forma independiente por lo que el Front End se encarga de crear un puente de comunicación entre ellos, permitiendo que los SIG puedan representar la información proveniente de los dispositivos móviles gestionados por los servidores AVL.

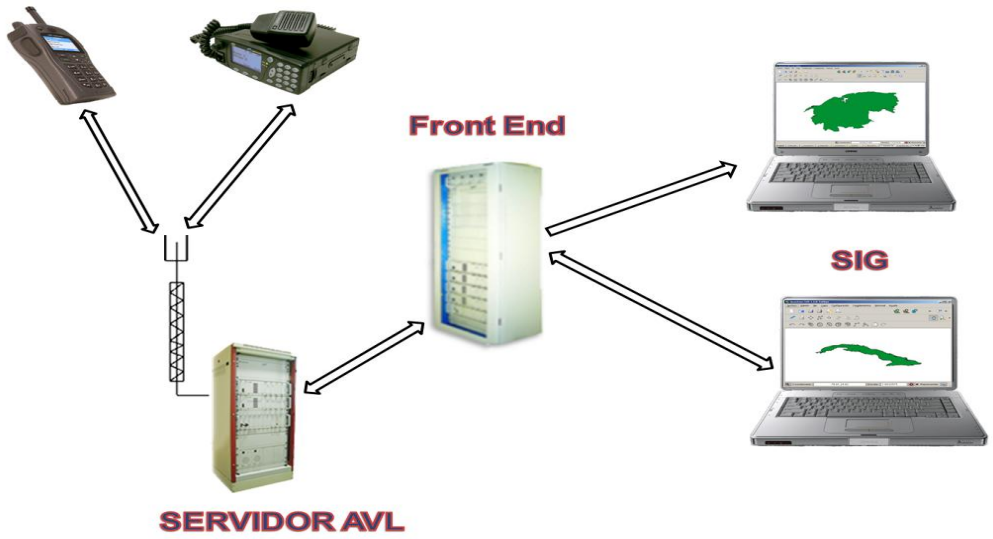

Figura 1: Infraestructura AVL.

1.2 Infraestructura TETRA.

La infraestructura TETRA básica está compuesta por un nodo central de control (SCN – System Control Node) y una serie de estaciones base (SBS – Site Base Station) que son el enlace entre los dispositivos móviles y la infraestructura mediante radiofrecuencia, tal como puede verse en la siguiente figura. [1]

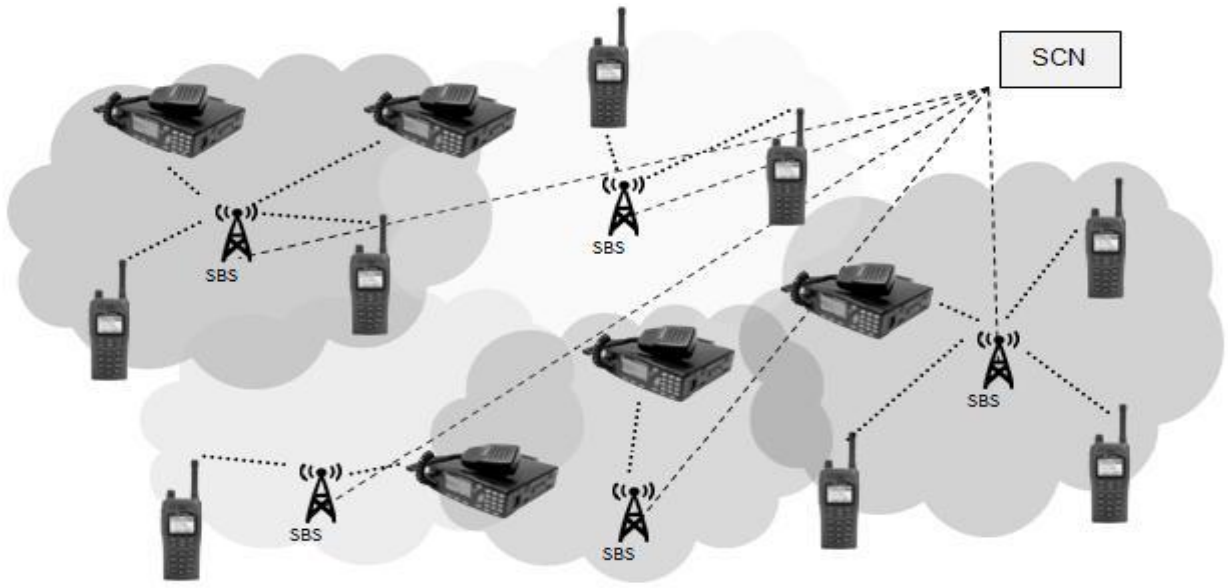

Figura 2: Esquema de Infraestructura TETRA. [1]

Cada terminal de radio se encuentra bajo la cobertura de una o varias portadoras y está registrado en la red. Cada SBS transmite información de broadcast a todos los terminales registrados indicando los servicios ofrecidos e indicando también el número de células vecinas y su portadora para que los terminales realicen una posible reelección sin perder registro en la red. Las SBS están compuestas por varios módulos entre ellos el servidor AVL NEBULA que gestiona la información geográfica de los terminales. [1]

1.2.1 Servidor AVL NEBULA

El servidor AVL NEBULA se encarga de gestionar la posición geográfica de los dispositivos móviles a través de la estación base. Este realiza la comunicación con la aplicación externa por puerto socket usando dos direcciones IP, una para la conexión estándar y otra de réplica en caso de fallo. El servidor

AVL NEBULA permite una sola sesión donde la aplicación externa que lo maneja debe usar dos puertos para establecer la comunicación, uno para los comandos y otros para los datos.

Figura 3: Comunicación entre el servidor AVL NEBULA y la aplicación externa.

Cada comando consiste en un texto ASCII, seguido por una o más líneas en blanco indicado el fin de mensaje. Los comandos que se envían a través del puerto de comandos los inicia la aplicación externa hacia el servidor y este siempre le responde con otro comando de aceptación dándole la información solicitada, si el comando enviado posee algún error el servidor lo informará. Los comandos que se envían a través del puerto de datos los envía solamente el servidor para notificar algún suceso a la aplicación externa.

1.2.2 Puerto Socket.

Un socket (enchufe), es un método para la comunicación entre un programa del cliente y un programa del servidor en una red. Un socket se define como el punto final en una conexión. 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). [2]
Un socket es también una dirección de Internet, combinando una dirección IP (la dirección numérica única de cuatro partes que identifica a un ordenador particular en Internet) y un número de puerto (el número que identifica una aplicación de Internet particular, como FTP, Gopher, o WWW). [2]

1.3 Sistemas de Información Geográfica

Un Sistema de Información Geográfica (SIG) es un conjunto de tecnologías, datos, métodos y recursos humanos que están diseñados para recolectar, almacenar, transformar, presentar y analizar espacialmente datos georeferenciados, creando un modelo digital del mundo real. [3]

Resulta evidente la importancia que tiene la disponibilidad de información objetiva, precisa, confiable y oportuna para la toma de decisiones. En el ámbito público, el desarrollo de proyectos sociales y económicos, las estrategias de prevención, la identificación de recursos, los estudios de impacto, en suma, la construcción de políticas públicas de todo orden, requiere de herramientas tecnológicas adecuadas. [3]

En este sentido los Sistemas de Información Geográfica procesan datos espaciales provenientes de diferentes fuentes que se integran en un mismo ambiente. Permiten la superposición de diferentes objetos (ríos, rutas, áreas censales, etc.) con variables distribuidas espacialmente referidas a dimensiones físicas (temperatura, propiedad del suelo, etc.), e indicadores estadísticos, sociales y económicos (tasas de mortalidad infantil, cantidad de personas con educación universitaria, cantidad de viviendas deficitarias, etc.). [3]

La flexibilidad del SIG en el tratamiento de la información y análisis a través la exploración visual territorial, ha transformado esta herramienta tecnológica en un instrumento de creación y contraste de hipótesis o en la aproximación al conocimiento de nuevos fenómenos ambientales y socioeconómicos. [3]

1.3.1 Formatos para representación de información GPS.

Los sistemas de información geográfica actuales se destacan por la variedad de formatos que pueden representar, estos formatos son capaces de contener varios tipos de información y soportar desde un simple punto hasta complejas construcciones en tres dimensiones.

Formato KML.

KML es un formato de archivo que se utiliza para mostrar información geográfica en navegadores terrestres como Google Earth, Google Maps y Google Maps para móviles. KML utiliza una estructura basada en etiquetas con atributos y elementos anidados y está basado en el estándar XML. [4]

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>Simple placemark</name>
    <description>Attached to the ground. Intelligently places itself
       at the height of the underlying terrain.</description>
    <Point>
      <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

Figura 4: Ejemplo de formato KML. [4]

Servicio TIG.

Los sistemas de información geográfica generalmente requieren representar la información en tiempo real, por lo que la lectura desde ficheros no es la más óptima debido a la demora que esta puede ocasionar, esto implica que se usen otros mecanismos como son las conexiones mediante puerto socket que posibilita una mayor rapidez con la transferencia de la información, además de permitir mecanismos adicionales de seguridad. Para compartir la información el equipo de desarrollo diseñó el servicio TIG, que permite a las aplicaciones clientes conectarse por puerto socket y recibir las tramas TIG (ver anexo 1).

1.4 Sistemas de posicionamiento vehicular sobre redes TETRA.

1.4.1 CeCo-SEC

Para Seguridad Pública, Emergencias y Bomberos

Optimizada para seguridad pública, la serie CeCo-SEC aprovecha la experiencia de Teltronic en el suministro de soluciones completas para agencias de Seguridad Pública, Emergencias y Bomberos durante los últimos 30 años. CeCo-SEC está enfocado a la gestión de recursos, el mantenimiento de llamadas y la adquisición de datos de incidentes; siendo una de sus principales ventajas la integración transparente y automatizada de todas estas funciones. [5]

CeCo-TRANS

Centros de Gestión de Transporte Público

El transporte público es un segmento clave en la sociedad actual. Su crecimiento continuo en las zonas urbanas está creando nuevos retos para las necesidades de la comunicación en este sector. El incremento de la seguridad y la eficacia de estos sistemas son fundamentales para satisfacer las necesidades del sector del transporte. [5]

Tanto el despacho vía línea de llamadas como la Localización Automática de Vehículos (AVL) contribuye a mejorar la eficacia de los sistemas de transporte público. La serie CeCo-TRANS está orientada específicamente a la gestión y control de flotas de transporte a través de comunicaciones vía radio. [5]

CeCo-CAD

Despachador de Aplicación General

CeCo-CAD es una solución de despacho vía línea totalmente integrada con dichas infraestructuras a través de interfaces Ethernet/IP, y que permite un control total de la operativa de la red. El interfaz de usuario es altamente configurable y permite la configuración de operaciones para una amplia gama de entornos profesionales como compañías de servicios públicos, compañías de gas y petrolíferas, gobierno, fuerzas armadas, etc. [5]

CeCo-911/112

Para Centros de llamadas 911 y 112

La potente arquitectura de la serie CeCoCo de Teltronic permite una escalabilidad con bajo coste desde centros con pocos operadores hasta complejos centros de control regionales. CeCo-911/112 está diseñado para posibilitar el intercambio de información entre distintas agencias y organizaciones de manera eficiente, permitiendo de esta manera una respuesta rápida, precisa y coordinada a situaciones de emergencia. [5]

1.4.2 MOTOLOCATOR de la empresa Motorola

MOTOLOCATOR es una aplicación de servidor que recibe y almacena datos de localización desde variados dispositivos compatibles con GPS. Traduce los múltiples y variados protocolos GPS que se utilizan actualmente y almacena esta información en una base de datos Oracle o Servidor Microsoft SQL. Luego hace que esta información esté disponible a través de una interfaz de Servicios Web estándar de la industria. [6]

Los datos de localización pueden ser visualizados a medida que se obtienen o pueden ser guardados para ser visualizados en el futuro para determinar la localización en la que se encontraba determinado dispositivo en un momento dado. [6]

Ofrece distintas opciones para la administración de dispositivos, incluida la posibilidad de establecer la velocidad a la que el dispositivo reporta su ubicación según distintos parámetros (tiempo, distancia y ciertos disparadores del dispositivo: encendido, batería baja, activación de botón de emergencia, entre otros). Ello le permite optimizar el ancho de banda, a la vez que garantiza que las actualizaciones de localización se lleven a cabo de manera oportuna. [6]

Muchos radios TETRA modernos cuentan con la funcionalidad GPS integrada, con actualizaciones enviadas como mensajes de datos cortos. MOTOLOCATOR admite muchos de estos radios, incluidos dispositivos Motorola, Cleartone y Sepura, así como también cualquier otro dispositivo que cumpla con el nuevo estándar ETSI LIP. Radios más antiguos que no cuentan con GPS integrado se pueden conectar a una unidad GPS independiente, ofreciendo el mismo nivel de funcionalidad. [6]

1.5 Propuesta para la solución.

En la fase de planificación y estimación del proyecto el cliente entregó una serie de datos de partida que se convirtieron en funcionalidades que debía cumplir el sistema. La necesidad del cliente y la dirección del proyecto de poder ver y monitorizar el desarrollo del mismo en espacios de tiempos relativamente cortos, además de la complejidad que resalta una solución de este tipo para el sector de la radiocomunicaciones, llevó al grupo del proyecto a realizar un estudio de qué metodología de desarrollo de software se adaptaba a este entorno, arrojando que la metodología FDD - *Feature Driven Development / Desarrollo Basado en Funcionalidades*- se adaptaba a las necesidades del equipo de desarrollo. También se acordó que las herramientas para el desarrollo deberían cumplir con el principio de software libre. Como herramienta CASE -*Computer Aided Software Engineering / Ingeniería de Software Asistida por Computadora*- se propone utilizar StarUML, partiendo de la premisa que los diagramas generados por esta herramienta gratuita están dentro de las especificadas por el cliente para la generación de diagramas. Además se seleccionó el framework QT 4.7 y el lenguaje C++ para el desarrollo del sistema.

1.6 Lenguajes y tecnologías utilizadas.

1.6.1 Qt Framework

Qt es un framework multiplataforma, que se utiliza para el desarrollo de aplicaciones, está escrito en C++, sin embargo, es posible utilizar Qt con otros lenguajes a través de enlaces. Existen bindings de Qt para lenguajes como C#, PHP, Python, y Ruby, entre otros. [7]

En un principio, Qt sólo ofrecía bibliotecas de código para la creación de interfaces gráficas de usuario. Ahora existen bibliotecas para muchas cosas más, como: Bases de datos, XML, multimedia, comunicación en red, OpenGL, etc. [7]

Qt extiende el lenguaje C++, a través de macros y meta información, mientras se mantiene apegado a él. Algunas características que agrega Qt a C++ son: Bucle foreach, sentencia forever e introspección. [7]

Qt está compuesto por una serie de módulos que proveen funcionalidad específica a través de una biblioteca de clases multiplataforma. Aunque también existen algunos módulos específicos para cada

plataforma, por ejemplo, QtDBUS para comunicación entre procesos, exclusiva de Unix o QtAxContainer y QtAxServer para construir y utilizar componentes ActiveX, exclusiva de Windows. [7]

Algunos de los módulos que forman Qt son: [7]

- Bases de Datos - Qt SQL
- Core - Qt Core
- Comunicación en red - Qt Network
- Interfaz Gráfica de usuario - Qt GUI
- Multimedia - Phonon, Qt Multimedia
- Quick - Qt Declarative, QML
- Webkit - Qt Webkit
- XML - Qt XML

Qt está disponible bajo 3 diferentes licencias: [7]

- **GPL** Aplicación de código abierto, los cambios realizados al código fuente de Qt deben ser compartidos con la comunidad.
- **LGPL** Es posible crear aplicaciones de código cerrado, los cambios realizados al código fuente de Qt deben ser compartidos con la comunidad.
- **Comercial** Es posible crear aplicaciones de código cerrado, los cambios realizados al código fuente de Qt pueden mantenerse cerrados.

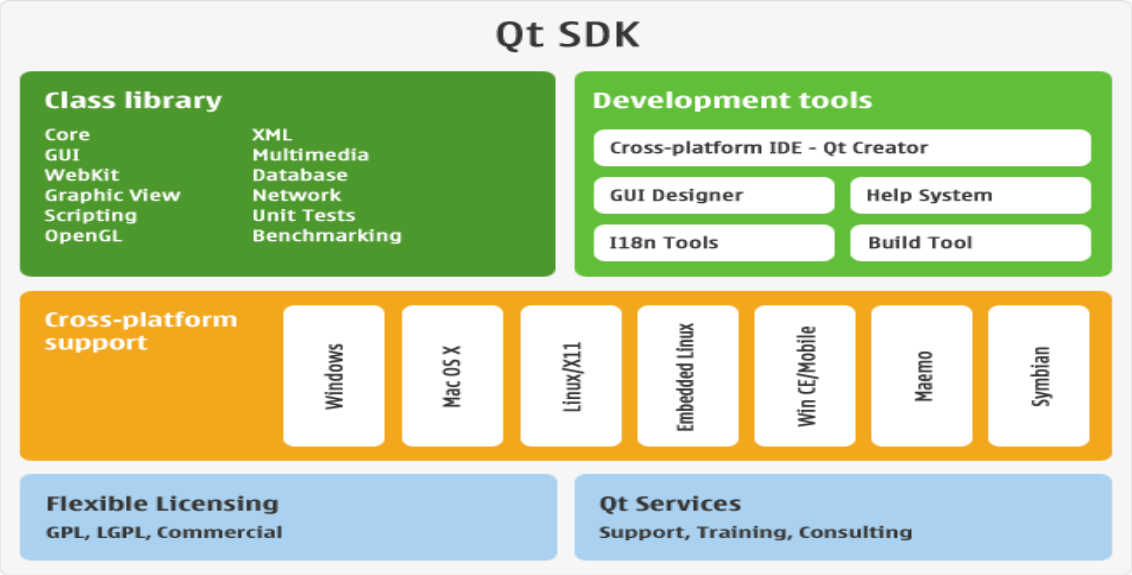

Figura 5: Estructura del framework Qt.

1.6.2 El lenguaje de programación C++.

C++ es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones. El C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original. [8]

El C++ es a la vez un lenguaje procedural (orientado a algoritmos) y orientado a objetos. Como lenguaje procedural se asemeja al C y es compatible con él, aunque ya se ha dicho que presenta ciertas ventajas (las modificaciones menores, que se verán a continuación). Como lenguaje orientado a objetos se basa en una filosofía completamente diferente, que exige del programador un completo cambio de mentalidad. Las características propias de la Programación Orientada a Objetos (Object Oriented Programming, u OOP) de C++ son modificaciones mayores que sí que cambian radicalmente su naturaleza. [8]

1.7 Metodología de desarrollo de software y lenguaje de modelado.

En todo desarrollo de aplicaciones la utilización de una metodología de desarrollo de software es el punto clave del éxito para lograr un trabajo organizado y eficiente. Con el uso de una metodología el equipo de desarrollo debe ver una mejora en la producción y en la calidad respecto a trabajos anteriores. Existen dos corrientes entre estas, las denominadas “pesadas” y las “ligeras” o “ágiles” nombradas así por el proceso de documentación que estas generan. A continuación se describe las características de la metodología utilizada en el desarrollo del sistema.

1.7.1 Desarrollo Basado en Funcionalidades (FDD).

Desarrollo basado en funcionalidades / Feature Driven Development (FDD) es una metodología ágil para el desarrollo de sistemas a corto plazo. Se basa en un proceso de iteraciones cortas de aproximadamente dos semanas. Estas iteraciones se deciden en base a las funcionalidades que son pequeñas partes del software con significado para el cliente. [9]

A diferencia de otras metodologías ágiles no cubre todo el ciclo de vida sino sólo las fases de diseño y construcción y se considera adecuado para proyectos mayores y de misión crítica. FDD no requiere un modelo específico de proceso y se complementa con otras metodologías. Enfatiza cuestiones de calidad y define claramente entregas tangibles y formas de evaluación del progreso. [9]

Los principios de FDD son pocos y simples: [9]

- Se requiere un sistema para construir sistemas si se pretende escalar a proyectos grandes.
- Un proceso simple y bien definido trabaja mejor.
- Los pasos de un proceso deben ser lógicos y su mérito inmediatamente obvio para cada miembro del equipo.
- Vanagloriarse del proceso puede impedir el trabajo real.
- Los buenos procesos van hasta el fondo del asunto, de modo que los miembros del equipo se puedan concentrar en los resultados.
- Los ciclos cortos, iterativos, orientados por funcionalidades (features) son mejores.

Figura 6: Proceso FDD. [9]

Desarrollo de un modelo global: Cuando comienza el desarrollo, los expertos del dominio están al tanto de la visión, el contexto y los requerimientos del sistema a construir. Se divide el dominio global en áreas que son analizadas detalladamente. Los desarrolladores construyen un diagrama de clases o de objetos por cada área y se construye un modelo global del sistema. [9]

Construcción de una lista de funcionalidades: Una funcionalidad es un ítem útil a los ojos del cliente. Se elabora una lista de funcionalidades que resuma la funcionalidad general del sistema; la lista es elaborada por los desarrolladores y es evaluada por el cliente y se divide en subconjuntos según la afinidad y la dependencia de las funcionalidades, la lista es finalmente revisada por los usuarios y los responsables para su validación y aprobación. [9]

Planeación por funcionalidad: Se procede a ordenar los conjuntos de funcionalidades conforme a su prioridad y dependencia, y se asigna a los programadores jefes. [9]

Diseño por funcionalidad y Construcción por funcionalidad: Se selecciona un conjunto de funcionalidades de la lista, se procede a diseñar y construir la funcionalidad mediante un proceso iterativo. Una iteración puede tomar de unos pocos días a un máximo de dos semanas. El proceso iterativo incluye inspección de diseño, codificación, pruebas unitarias, integración e inspección de código. [9]

1.7.2 Leguaje Unificado de Modelado (UML).

El Lenguaje Unificado de Modelado (UML por sus siglas en inglés) es un lenguaje para la visualización, especificación y documentación de software, por lo que resulta independiente del método que se utilice para el desarrollo. No es un método sino una notación, pues no especifica un proceso; lo que se hace es describir el resultado de alguna etapa del desarrollo de un sistema mediante una serie de diagramas. [10]

UML es un lenguaje que permite la modelación de sistemas con tecnología orientada a objetos. También intenta solucionar el problema de propiedad de código que se da con los desarrolladores, al implementar un lenguaje de modelado común para todos los desarrollos se crea una documentación también común, que cualquier desarrollador con conocimientos de UML será capaz de entender, independientemente del lenguaje utilizado para el desarrollo.

La idea general de UML, así como de todas las notaciones de modelado orientados a objetos, es centrarse más en los objetos que en los procesos o algoritmos. [10]

En términos muy generales UML define una notación que se expresa con diagramas. Para mostrar las diferentes perspectivas del modelado, UML define 9 tipos de diagramas: [10]

- Diagramas de casos de uso
- Diagramas de clases
- Diagramas de componentes
- Diagramas de despliegue
- Diagramas de objetos
- Diagramas de colaboración
- Diagramas de estados y transiciones
- Diagramas de actividades

1.8 Herramientas usadas.

1.8.1 StarUML.

StarUML es una herramienta para el modelamiento de software basado en los estándares UML (Unified Modeling Language) y MDA (Model Driven Arquitecture), que en un principio era un producto comercial y que hace cerca de un año pasó de ser un proyecto comercial (anteriormente llamado plastic) a uno de licencia abierta GNU/GPL. [11]

El software heredó todas las características de la versión comercial y poco a poco ha ido mejorando sus características, entre las cuales se encuentran: [11]

- Soporte completo al diseño UML mediante el uso de.
  – Diagrama de casos de uso
  – Diagrama de clase
  – Diagrama de secuencia
  – Diagrama de colaboración.
  – Diagrama de estados
  – Diagrama de actividad.
  – Diagrama de componentes
  – Diagrama de despliegue.
  – Diagrama de composición estructural (UML 2.0)

- Definir elementos propios para los diagramas, que no necesariamente pertenezcan al estándar de UML.
- La capacidad de generar código a partir de los diagramas y viceversa, actualmente funcionando para los lenguajes C++, C# y java.
- Generar documentación en formatos Word, Excel y PowerPoint sobre los diagramas.
- Patrones GoF (Gang of Four), EJB (Enterprise JavaBeans) y personalizados.
- Plantillas de proyectos.
- Posibilidad de crear plugins para el programa.

1.8.2 Qt Creator

Qt es una biblioteca de software que desarrolla Nokia para crear interfaces gráficas de usuario. Aplicaciones como: Google Earth, Skype, Adobe Photoshop Album y VirtualBox entre muchas otras, hacen uso de esta. [12]

Para los desarrolladores, Nokia ofrece Qt Creator, un entorno de desarrollo (IDE) multiplataforma muy completo. [12]

Principales características de Qt Creator: [12]

- Posee un avanzado editor de código C++.
- Además soporta los lenguajes: C#/.NET Languages (Mono), Python: PyQt y PySide, Ada, Pascal, Perl, PHP y Ruby.
- Posee también una GUI integrada y diseñador de formularios.
- Herramienta para proyectos y administración.
- Ayuda sensible al contexto integrada.
- Depurador visual.
- Resaltado y auto-completado de código.
- Soporte para refactorización de código.

Qt Creator es distribuido bajo tres tipos de licencias: Qt Commercial Developer License, Qt GNU LGPL v. 2.1, Qt GNU GPL v. 3.0 y está disponible para las plataformas: Linux, Mac OSX; Windows, Windows CE, Symbian y Maemo. [12]

Conclusiones.

En este capítulo se ha realizado una investigación relacionada con las soluciones que gestionan la información de los servidores AVL y el manejo de esta con los SIG. También se determinó la metodología de desarrollo a utilizar, así como los lenguajes, tecnologías y herramientas que serán empleados en el sistema.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

Introducción.

En este capítulo se describe la solución al problema a resolver. Se detalla la estructura del sistema y el modelo de dominio. Además se enumera los requerimientos funcionales y no funcionales del sistema. Así como una descripción de los casos de uso del sistema.

2.1 Objeto de automatización.

En la actualidad la empresa Teltronic posee una solución que le permite realizar el control y seguimiento de los dispositivos móviles, pero no tiene la capacidad de integrarse con otros SIG. Esta aplicación tiene como objetivo establecer una comunicación con el servidor AVL NEBULA y darle la posibilidad al usuario de exportar la información de los dispositivos a formato KML, que es muy usado por los SIG actuales. Además de ofrecer un mecanismo adicional para compartir la información de una forma más rápida usando el servicio TIG.

2.2 Descripción de la solución propuesta.

El Front End tiene la función de conectarse al servidor AVL NEBULA a través del puerto socket y mediante los comandos de comunicación (ver anexo 2) obtener la información geográfica de cada uno de los dispositivos móviles en tiempo real. Además debe configurar el modo operacional del servidor en cíclico o secuencial, así como el formato en que se recibirá la información geográfica, ya sea en NMEA o LIP. Para guardar los datos obtenidos se crean tareas de tipo KML, que son las que permiten el seguimiento de los dispositivos móviles desde los SIG que soportan este formato. Además se pueden crear tareas TIG para compartir la información con los SIG que consuman de este servicio.

2.3 Estructura del sistema.

En el mundo actual los sistemas informáticos se desarrollan con una estructura pensada para resolver las necesidades del cliente y los problemas que puedan surgir en el sistema a la hora de efectuar cambios. Para esto se diseñó con una estructura flexible, que pueda crecer y actualizarse a medida que sea

necesario. Esta estructura está compuesta por dos partes fundamentales, el núcleo y los plugins. Entre las funciones del núcleo está la de cargar todos los plugins disponibles, para esto el núcleo busca en tiempo de ejecución en el directorio Plugins, verificando que cada plugin esté correctamente implementado y comprobando a qué tipo pertenece para poder adicionarlos y de esta forma extender las funcionalidades del sistema.

Los plugins se dividen según su función en dos partes, los servidores y las tareas. Los servidores deben implementar una interfaz que ofrece el sistema para que este pueda reconocerlos, siendo su función establecer comunicación con servidores AVL y obtener la información proveniente de estos para proporcionarla al sistema de una forma predeterminada, de esta manera se crea una capa de abstracción donde el sistema puede manejar cada servidor sin importar el origen de datos que este pueda usar. Las tareas también deben implementar una interfaz y su función es ofrecer vías para que la información proveniente del sistema pueda ser accedida y manipulada por los SIG, de igual forma maneja cada tarea sin importar cómo fue implementada ni la manera que va a compartir la información. Cada plugin está compuesto por dos librerías dinámicas, una para la vista y otra para el modelo debido a que el sistema implementa el patrón arquitectónico MVC -*Model View Controller / Modelo Vista Controlador*- donde el núcleo hace el papel de controlador. El uso de esta estructura fomenta la escalabilidad, debido a que el sistema mejora funcionalmente después de haberle añadido plugins, incrementando su capacidad funcional. Además se logra un aumento en la productividad puesto que se puede involucrar todos los medios disponibles en el desarrollo del sistema sin crear dependencia entre sí y trabajando de manera paralela lo que implica que se reduzca considerablemente el ciclo de desarrollo del software. Otras de las ventajas que ofrece es la realización de pruebas ya que no se necesita probar todo el sistema sino solamente la parte del plugin que se desea verificar, permitiendo que cada prueba se pueda realizar de manera independiente.

La actualización y soporte fueron elementos de suma importancia que se tuvieron en cuenta para definir la estructura del sistema. Con esta se logra mejorar el proceso de actualización y soporte porque no se necesita compilar todo el código sino solamente la parte que se desea actualizar o arreglar, de igual manera en el proceso de comercialización le brinda al cliente la posibilidad de elegir las funcionalidades que desee y no el sistema completo.

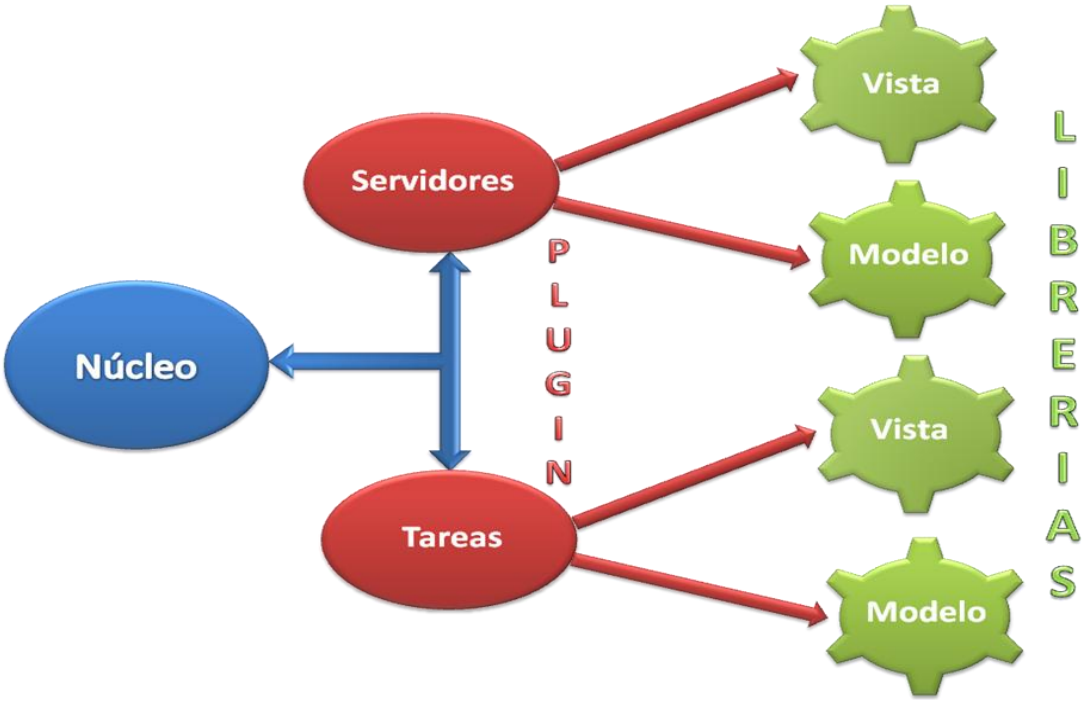

Figura 7: Estructura del sistema.

2.4 Modelo de dominio.

A continuación se detalla el modelo de dominio realizado para darle solución a las principales funcionalidades. Se especifica además con un lenguaje más comprensible tanto para los clientes como para los desarrolladores cada uno de los conceptos y sus relaciones.

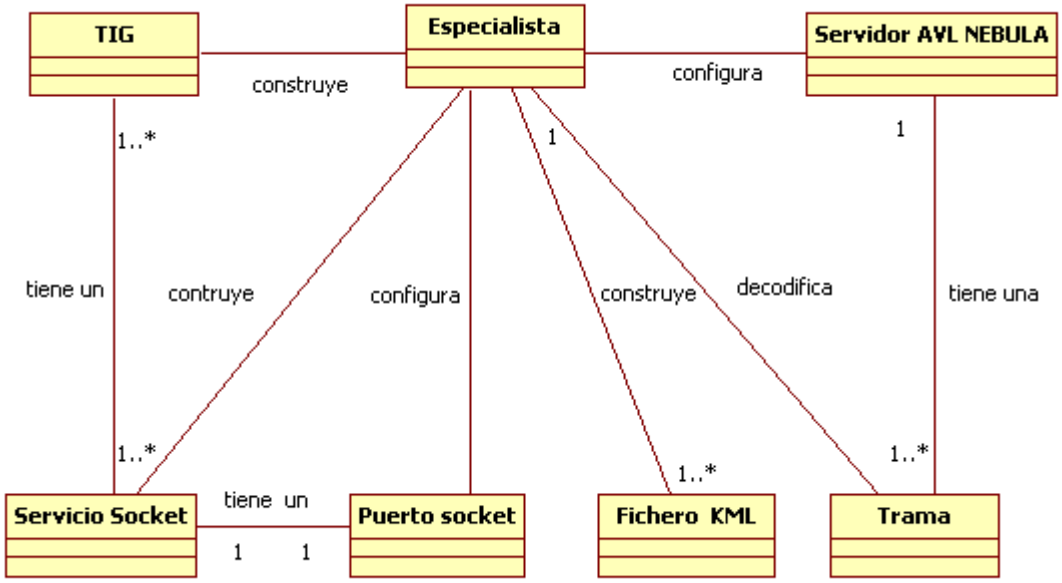

Figura 8: Modelo de Dominio.

Especialista: Especialista encargado de realizar la conexión con el servidor AVL NEBULA por puerto socket y de configurar el modo de funcionamiento del mismo. Además comparte la información recibida por medio de las tareas KML o TIG.

Servidor AVL NEBULA: Servidor que es configurado por el especialista para poder obtener la información contenida en él.

Puerto Socket: Configuración previa que se realiza al puerto socket por parte del especialista.

TIG: Formato de trama que se utiliza para almacenar la información geográfica.

Servicio Socket: Servicio que brinda la trama TIG mediante el puerto socket.

Fichero KML: Fichero que contiene la información obtenida del servidor AVL NEBULA en formato KML.

Trama: Contiene la información geográfica en formato NMEA (ver anexo 3) o LIP (ver anexo 4).

El Especialista para crear una conexión con el servidor AVL NEBULA configura los parámetros de conexión por puerto socket (IP, Puerto de Datos, Puerto de Comandos) y mediante los comandos de

comunicación puede configurar el modo de funcionamiento del servidor. Toda la información recibida desde el servidor puede ser compartida a los SIG mediante las tareas KML o TIG.

2.5 Especificación de las funcionalidades.

2.5.1 Requerimientos funcionales

Los requerimientos funcionales son condiciones o capacidades que el sistema debe cumplir. A continuación se enumeran los que han sido identificados.

RF 1. Gestionar Servidor AVL NEBULA.

RF 1.1. Crear Servidor AVL NEBULA.

- Servidor
- Puerto de Datos
- Puerto de Comandos

RF 1.2. Configurar Servidor AVL NEBULA.

- Modo Operacional
- Formato GPS

RF 2. Administrar Servidor AVL.

RF 2.1. Iniciar Servidor AVL
RF 2.2. Pausar Servidor AVL
RF 2.3. Detener Servidor AVL
RF 2.4. Eliminar Servidor AVL
RF 2.5. Activar Dispositivo
RF 2.6. Desactivar Dispositivo
RF 2.7. Iniciar Todos los Servidores AVL
RF 2.8. Pausar Todos los Servidores AVL

RF 2.9. Parar Todos los Servidores AVL

RF 2.10. Eliminar Todos los Servidores AVL

RF 3. Gestionar Tarea KML.

RF 3.1. Adicionar Tarea KML.

- Nombre
- Dirección
- Tiempo

RF 3.2. Configurar Tarea KML.

- Dirección
- Tiempo

RF 4. Gestionar Tarea TIG.

RF 4.1. Adicionar Tarea TIG.

- Nombre
- Puerto
- Cantidad

RF 4.2. Actualizar Tarea TIG.

- Cantidad

RF 5. Administrar Tarea

RF 5.1. Iniciar Tarea

RF 5.2. Detener Tarea

RF 5.3. Eliminar Tarea

RF 5.4. Activar Servidor

RF 5.5. Desactivar Servidor

RF 5.6. Activar Dispositivo

RF 5.7. Desactivar Dispositivo

RF 5.8. Iniciar Todas las Tareas

RF 5.9. Parar Todas las Tareas

RF 5.10. Eliminar Todas las Tareas

2.5.2 Requerimientos no funcionales.

En muchos casos los requerimientos no funcionales son fundamentales en el éxito del producto y normalmente están vinculados a requerimientos funcionales. Se han determinado los siguientes requisitos no funcionales.

Requerimientos de Apariencia o interfaz externa

- Interfaz con un diseño sencillo que contenga pocos gráficos, con vista a acelerar la velocidad de respuesta; sin dejar de ser amigable, legible, interactiva, fácil de usar, profesional, clara y sencilla.

Requerimientos de Hardware.

- Micro ≥ 1.6 GHz
- Memoria RAM ≥ 512 MB

Requerimientos de Rendimiento

- La aplicación debe ser eficiente.
- Se debe poder capturar la información del Servidor AVL NEBULA rápidamente.
- Se debe poder exportar la información geográfica rápidamente.

Requerimientos de Soporte

- El sistema debe ser de fácil instalación.
- El sistema debe estar bien documentado de forma tal que el tiempo de mantenimiento sea

mínimo en caso de necesitarse.

- El sistema debe ser fácil de actualizar.

Requerimientos de diseño

- El lenguaje de programación que se usará es C++.
- Para el diseño del sistema debe ser utilizada la metodología FDD, usando el lenguaje de modelación UML y como herramienta para llevarlo a cabo el StarUML.

Ayuda

- El sistema brindará a los usuarios una ayuda. De este modo cuando el usuario presente algún problema, pueda acudir a la misma.

Requerimientos de Seguridad:

- El sistema debe asegurar integridad y disponibilidad.

2.6 Modelo de Caso de Uso del Sistema.

2.6.1 Definición del actor del sistema a automatizar.

Actores	Justificación
Especialista	Es un usuario del sistema que puede conectarse y configurar el servidor AVL NEBULA. También se encarga de adicionar tareas y configurarlas para compartir la información con los SIG deseados.

2.6.2 Diagrama de casos de uso del sistema a automatizar.

Los diagramas de casos de uso se emplean para modelar la vista de casos de uso de un sistema. La mayoría de las veces, esto implica modelar el contexto del sistema, subsistema o clase, o el modelado de los requisitos de comportamiento de esos elementos.

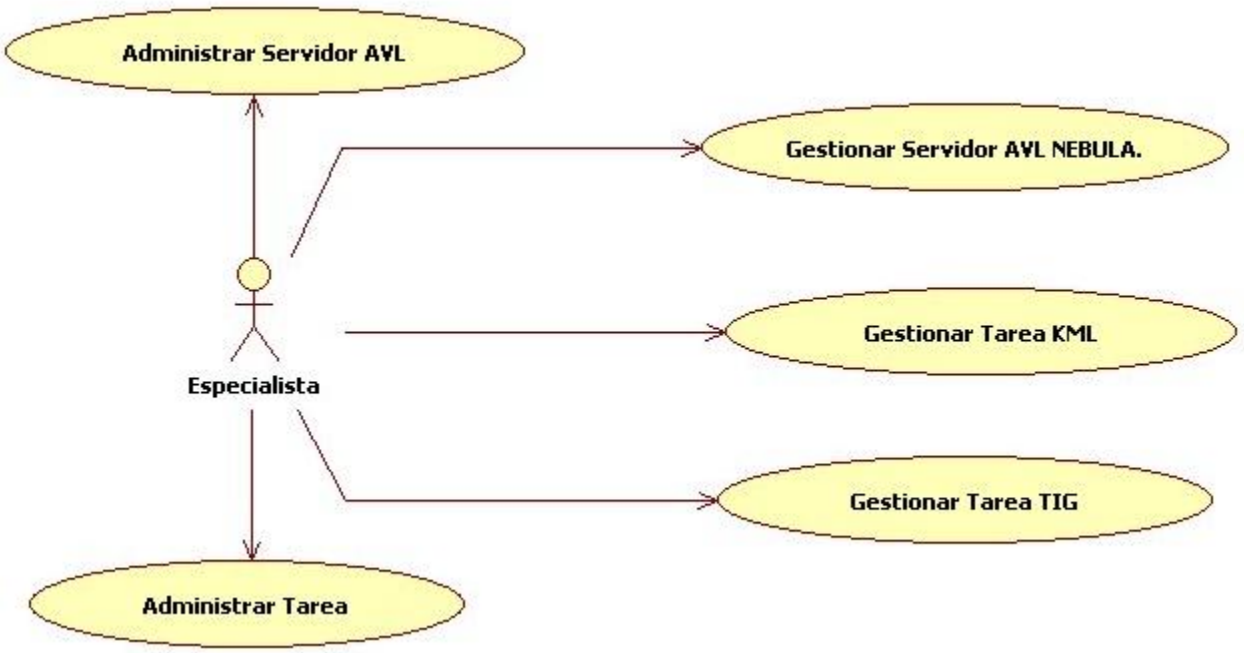

Figura 9: Diagrama de Caso de Uso de Sistema.

2.6.3 Descripción de los Casos de uso del Sistema.

En el Anexo 5 se encuentran las descripciones textuales de los casos de usos.

Conclusiones.

En el presente capítulo se ha tratado la descripción de la solución propuesta, así como cada uno de los requerimientos funcionales y no funcionales que se han podido identificar. Además de una descripción detallada de los casos de uso del sistema propuesto. También la representación del modelo de dominio así como una descripción detallada del mismo.

CAPÍTULO 3: DISEÑO DEL SISTEMA

Introducción.

En este capítulo se hace referencia a los aspectos correspondientes al diseño, en función de la propuesta del sistema y teniendo en cuenta las funcionalidades seleccionadas se confeccionan los diagramas de clases del diseño y los diagramas de interacción. Además se describen las clases del diseño más importantes en función de establecer la estructura del sistema. Se especifican los patrones de diseño y se realiza la propuesta de interfaz de usuario para el sistema.

3.1 Patrones utilizados.

Los patrones son estructuras de implementación que logran una finalidad determinada en un lenguaje de programación de alto nivel. En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Existen diferentes tipos de patrones: patrones arquitectónicos y patrones de diseño. Un patrón arquitectónico especifica un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes. Un patrón de diseño está relacionado con los aspectos del diseño de los subsistemas. Es una solución estándar para un problema común de programación y una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios.

3.1.1 Patrón de arquitectura Modelo Vista Controlador.

El patrón de arquitectura Modelo Vista Controlador separa los datos del sistema en tres capas fundamentales.

Modelo: Representación específica de la información con la cual el sistema opera. En resumen, el modelo se limita a lo relativo de la vista y su controlador facilitando las presentaciones visuales complejas. El sistema también puede operar con más datos no relativos a la presentación, haciendo uso integrado de otras lógicas de negocio y de datos afines con el sistema modelado. [13]

Vista: Presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario. [13]

Controlador: Responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y, probablemente, a la vista. [13]

El uso de este patrón permite que la vista y el modelo estén totalmente separados, lo que posibilita que ambos sistemas se puedan desarrollar independientes. Para lograr la comunicación entre ambas partes se usa el controlador que se encarga de manejar las peticiones realizadas a través de eventos, permitiendo mayor flexibilidad al no existir una asociación directa entre el controlador y las demás capas.

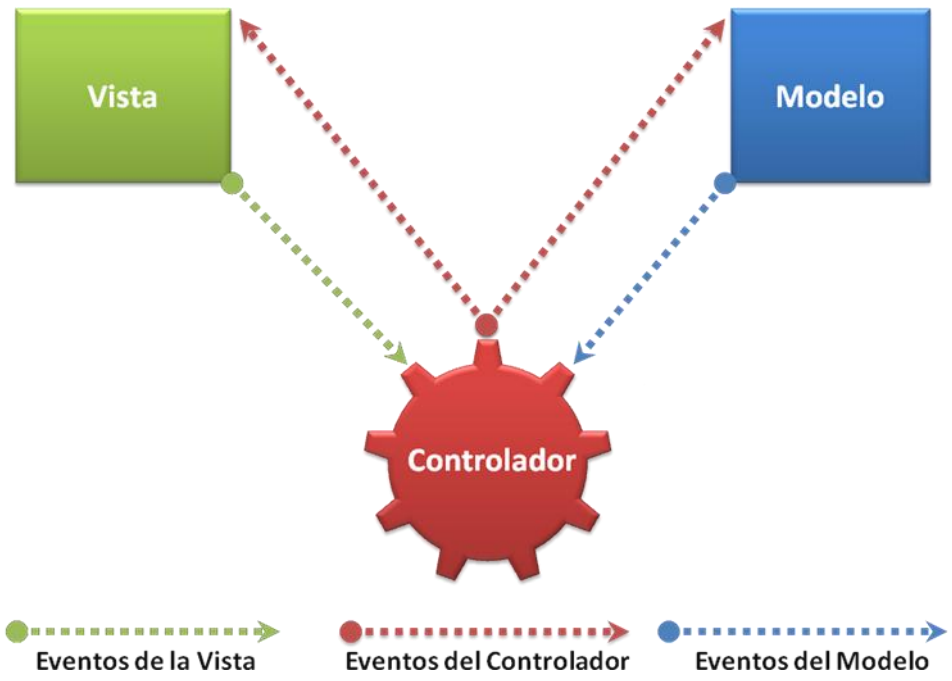

Figura 10: Diagrama del MVC del Sistema.

3.1.2 Patrones de Diseño GoF

Fábrica Abstracta

Proporcionar una interfaz para la creación de familias de objetos interdependientes o interrelacionados, sin sus clases concretas. [14]

En el sistema se usa este patrón para resolver el problema que surge cuando se necesitan agregar nuevas funcionalidades a través de plugins. Para esto se crea una interfaz para cada uno de los cuatro tipos de familias de objetos que se pretende crear donde cada nuevo plugins debe implementar la que le corresponde, lo que permite la creación de los mismos abstrayéndose de la implementación de sus clases.

Patrón Comando

Este patrón permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor real de la misma. Para ello se encapsula la petición como un objeto, con lo que además se facilita la parametrización de los métodos. [14]

Fue utilizado para permitir que más de dos controles pudieran ejecutar la misma acción. Por ejemplo, las acciones del menú principal pueden realizarse al seleccionar cualquier ícono de la barra de herramientas del contenedor principal. Para esto se define una acción en el sistema y se subscribe uno o varios interesados, después se adiciona a los contenedores de acciones, como por ejemplo menús y barras de tareas. De esta manera cuando un usuario presione sobre algún elemento en la barra de tarea o en el menú se dispara el mismo evento “triggered” y se ejecuta los métodos subscritos a este evento.

```
connect(ui->actionIniciar_Todos,SIGNAL(triggered()),this,SLOT(IniciarServidores()));
 menuServidores->addAction(ui->actionIniciar_Todos);
toolBarServidor>addAction(ui->actionIniciar_Todos);
```

Figura 11: Uso del patrón de diseño Comando.

Patrón Cadena de responsabilidad

Permite establecer una cadena de objetos receptores a través de los cuales se pasa una petición formulada por un objeto emisor. Cualquiera de los objetos receptores puede responder a la petición en función de un criterio establecido. [15]

Fue utilizado para permitir que cuando un evento sea lanzado desde el controlador hacia el modelo principal, este lo envíe hacia los modelos que poseen sin saber cuál le va dar respuesta ni que operación se pretende resolver.

Patrón Adaptador

Se utiliza para transformar una interfaz en otra, de tal modo que una clase que no pudiera utilizar la primera, haga uso de ella a través de la segunda. Convierte la interfaz de una clase en otra interfaz que el cliente espera. [16]

Fue utilizado para permitir que la vista de los plugins se comporte de acuerdo a las necesidades del modelo.

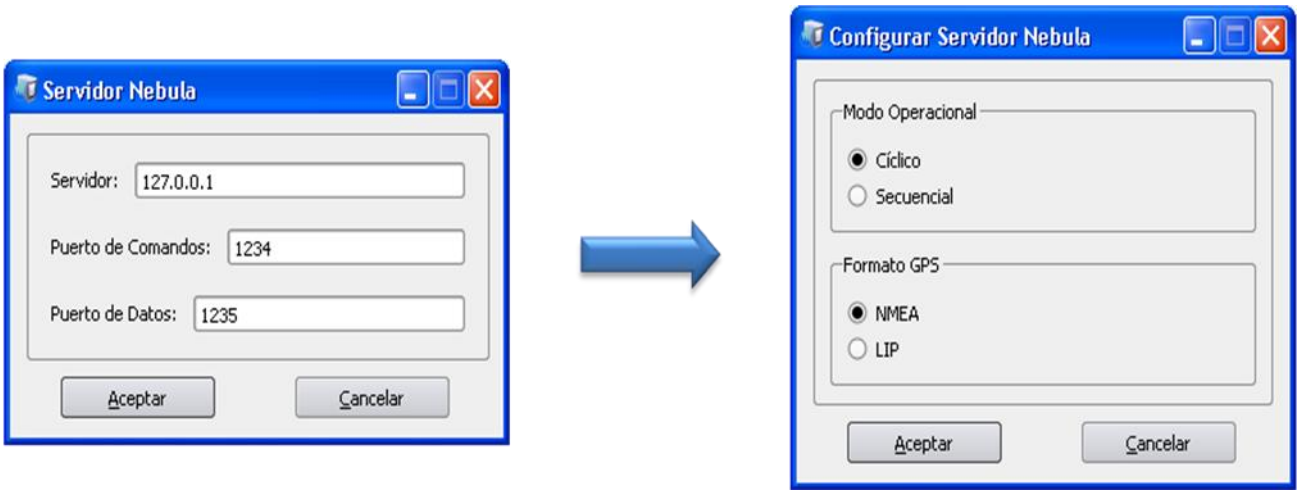

Figura 12: Uso del patrón de diseño Adaptador.

3.1.3 Patrones de Diseño GRASP

Experto

Se encarga de asignar una responsabilidad al experto en información, o sea, aquella clase que cuenta con la información necesaria para cumplir la responsabilidad. [17]

Fue utilizado para permitir que el modelo le asigne cada tipo de plugins solo las funciones que le corresponde a cada uno.

```
if(servidor)
{
connect(this,SIGNAL(MensajeEntrante(QString,QList<QObject*>)),servidor,SLOT(RecibirMensaje(QString,QList<QObject*>)));
connect(servidor,SIGNAL(EnviarMensaje(QString,QList<QObject*>)),this,SIGNAL(MensajeSaliente(QString,QList<QObject*>)));
connect(servidor,SIGNAL(Terminal(QString,QString,QString)),this,SIGNAL(Terminal(QString,QString,QString)));
}
if(tarea)
{
connect(this,SIGNAL(MensajeEntrante(QString,QList<QObject*>)),tarea,SLOT(RecibirMensaje(QString,QList<QObject*>)));
connect(tarea,SIGNAL(EnviarMensaje(QString,QList<QObject*>)),this,SIGNAL(MensajeSaliente(QString,QList<QObject*>)));
connect(this,SIGNAL(Terminal(QString,QString,QString)),tarea,SLOT(Terminal(QString,QString,QString)));
}
```

Figura 13: Uso del patrón de diseño Experto.

Controlador

Asigna la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase. [17]

Fue utilizado para controlar el flujo de eventos entre la vista y el modelo, permitiendo el intercambio de información a través del controlador.

```
connect(&vistaPrincipal,SIGNAL(CrearServidor(QString)),&modelo,SLOT(CrearServidor(QString)));
connect(&vistaPrincipal,SIGNAL(CrearTarea(QString)),&modelo,SLOT(CrearTarea(QString)));
connect(&vistaPrincipal,SIGNAL(Mensaje(QString,QList<QObject*>)),&modelo,SIGNAL(MensajeEntrante(QString,QList<QObject*>)));
connect(&modelo,SIGNAL(MensajeSaliente(QString,QList<QObject*>)),&vistaPrincipal,SIGNAL(MensajeEntrante(QString,QList<QObject*>)));
connect(&modelo,SIGNAL(Terminal(QString,QString,QString)),&vistaPrincipal,SLOT(RecibirTrama(QString,QString,QString)));
```

Figura 14: Uso del patrón de diseño Controlador.

Alta Cohesión

Tiene que ver con la forma en que agrupamos unidades de software en una unidad mayor; es decir los componentes de la unidad de software deben ser coherentes y estar relacionados lo mejor posible por ejemplo, la forma en que agrupamos clases en un paquete, la forma en que agrupamos métodos en una clase, etc. [18]

Se evidencia en el sistema con el uso de las interfaces que permiten agrupar las funcionalidades de acuerdo a su tipo, además de facilitar su reutilización.

Bajo Acoplamiento

Este término hace alusión al grado de dependencia que tienen dos unidades de software. El bajo acoplamiento en los métodos nos da la idea de lo dependientes que son entre sí, es decir el grado en que un método puede hacer su trabajo sin el otro. De igual forma si hablamos de paquetes el acoplamiento nos dará una idea de en qué medida el contenido de ese paquete puede hacer su trabajo sin el otro. [18]

Se evidencia en el sistema el bajo acoplamiento con el uso de la programación orientada a eventos, que permite que existan muy pocas dependencias directas entre las clases.

3.2 Diagramas de Paquetes del Diseño.

Los diagramas de paquetes permiten dividir un modelo en partes manejables mediante la agrupación de clases u otros paquetes. Estos diagramas se utilizan para ocultar detalles irrelevantes en el diagrama y

permiten dividir al sistema orientado a objetos organizándolo en subsistemas y detallando sus relaciones. Dentro de los paquetes se pueden representar los elementos incluidos, aunque no es habitual. Además estos diagramas contienen dos tipos de elementos: los paquetes y las dependencias.

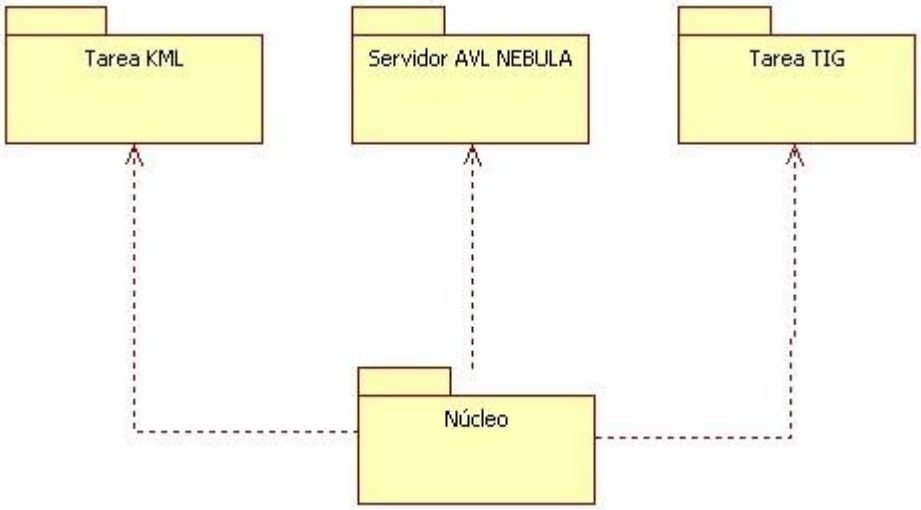

Figura 15: Diagrama de Paquetes del Diseño.

3.3 Diagrama de Clases del Diseño.

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro. Los diagramas de clases del diseño del sistema, se pueden encontrar en el Anexo 6 del presente documento.

3.4 Descripción de las clases fundamentales.

3.4.1 Clase *"CPrincipal".*

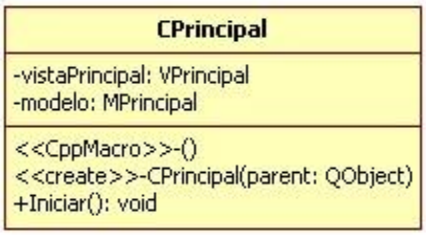

Figura 16: Clase "CPrincipal".

Propósito*:* El propósito de esta clase es controlar las peticiones que se crean con los eventos que se disparan entre la vista y el modelo, permitiendo que ambas partes puedan comunicarse aunque sean totalmente independientes.

3.4.2 Clase "MPrincipal".

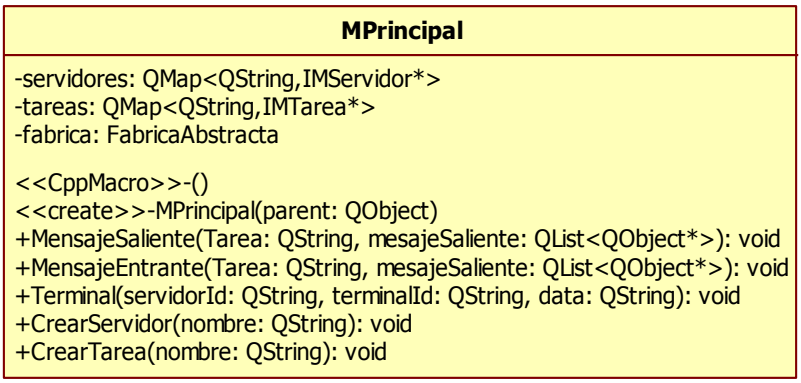

Figura 17: Clase "MPrincipal".

Propósito*:* El propósito de esta clase es atender y enviar peticiones al controlar y a su vez controlar todos los eventos en los modelos, tanto para servidores como para tareas. Para ello atiende las peticiones de la vista que son enviadas a través del controlador y estas pueden ser para crear modelos o para hacer

alguna petición a algún modelo específico. Otra de las funciones es distribuir los eventos que se lanzan en los modelos hacia las vistas por el controlador.

3.4.3 Clase “FabricaAbstracta”.

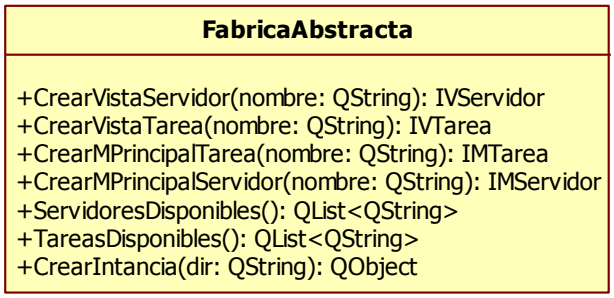

Figura 18: Clase “FabricaAbstracta”.

Propósito*:* El propósito de esta clase es cargar los plugins de las tareas y de los servidores según el tipo de petición, ya sea para crear una vista o un modelo, para ellos crea instancias del plugin devolviéndole un objeto de la interfaz, abstrayéndose de su implementación. Otra de sus funciones es verificar que estén correctamente implementados para poder usarse en el sistema.

3.4.4 Clase “IMTarea”.

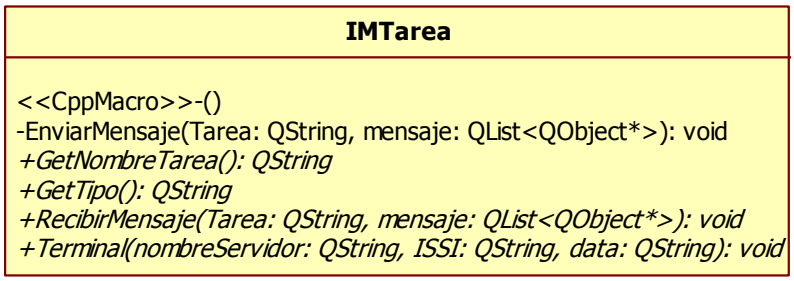

Figura 19: Clase “IMTarea”.

Propósito*:* El propósito de esta clase es que el sistema pueda cargar la librería dinámica que representa el modelo de una tarea en un plugin, sin importar como fue implementado. Permite además que se puedan crear librerías dinámicas para el modelo de las tareas.

3.4.5 Clase "IMServidor".

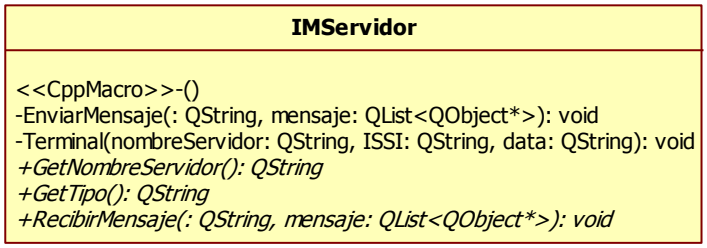

Figura 20: Clase "IMServidor".

Propósito*:* El propósito de esta clase es que el sistema pueda cargar la librería dinámica que representa el modelo de un servidor AVL en un plugin, sin importar como fue implementado. Permite además que se puedan crear librerías dinámicas para el modelo de los servidores.

3.4.6 Clase "IVTarea".

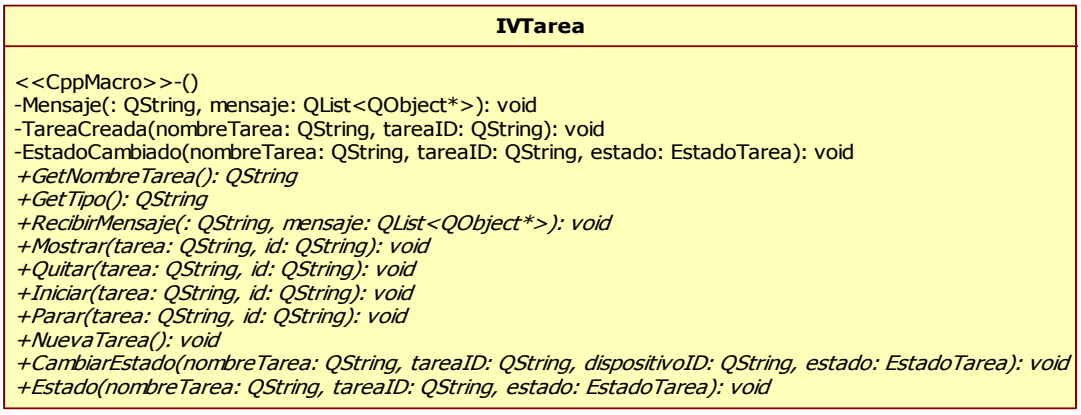

Figura 21: Clase "IVTarea".

Propósito*:* El propósito de esta clase es que el sistema pueda cargar la librería dinámica que representa la vista de una tarea en un plugin, sin importar como fue implementado. Permite además que se puedan crear librerías dinámicas para la vista de las tareas.

3.4.7 Clase “IVServidor”.

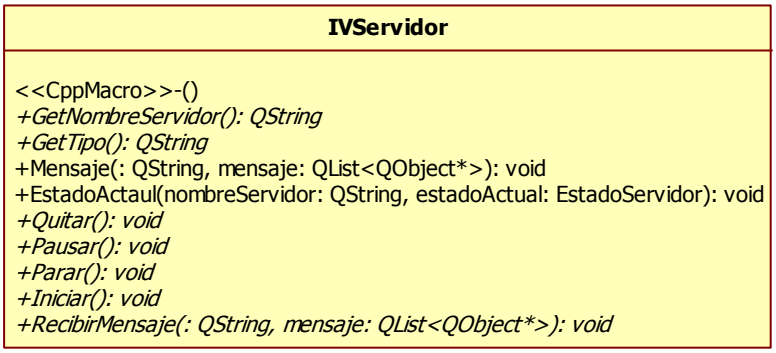

Figura 22: Clase “IVServidor”.

Propósito*:* El propósito de esta clase es que el sistema pueda cargar la librería dinámica que representa la vista de un servidor AVL en un plugin, sin importar como fue implementado. Permite además que se puedan crear librerías dinámicas para la vista de los servidores.

3.4.8 Clase “VPrincipal”.

VPrincipal

<<CppMacro>>-()
<<create>>-VPrincipal(parent: QWidget)
<<destroy>>-VPrincipal()
+Mensaje(: QString, mensajeSalida: QList<QObject*>): void
+MensajeEntrante(: QString, mensajeEntrante: QList<QObject*>): void
+ActualizarIdentificador(serv: QString, id: QString, activo: bool): void
+CrearServidor(serv: QString): void
+CrearTarea(tarea: QString): void
+Cargando(cargaId: QString): void
+CerraTodos(): void
+CerraPausar(): void
-RecibirMensaje(: QString, mensajeEntrante: QList<QObject*>): void
-RecibirTrama(servidorId: QString, dispositivoId: QString, datos: QString): void
-ActualizarVisual(: QModelIndex): void
-ActualizarVisualTarea(: QModelIndex): void
-ActualizarEstado(: QString, : EstadoServidor): void
-CrearNuevaTarea(tarea: QString, id: QString): void
-ActualizarDispositivo(serId: QString, dispId: QString, activa: bool): void
-ActualizarEstadoTarea(serId: QString, dispId: QString, estado: EstadoTarea): void
-IniciarServidores(): void
-ParararServidores(): void
-PausarServidores(): void
-QuitarServidores(): void
-IniciarTareas(): void
-ParararTareas(): void
-QuitarTareas(): void
-ActulizarTareas(): void
-ActulizarActions(): void
-ActulizarTarea(tareaActualizar: QString): void
-Mostrar(): void
-Quitar(): void
-Iniciar(): void
-Pausar(): void
-Parar(): void
-Activar(): void
-Desactivar(): void
-MostrarTarea(): void
-QuitarTarea(): void
-IniciarTarea(): void
-ParararTarea(): void
-ActivarDispositivo(): void
-DesactivarDispositivo(): void
-AddServidor(action: QAction): void
-AddTarea(action: QAction): void
-AdicionarNombrePlugin(): void
-CargarPlugins(): void
-Ayuda(action: QAction): void
-CrearAyuda(): void
#eventFilter(: QObject, : QEvent): bool

Figura 23: Clase “VPrincipal”.

Propósito*:* El propósito de esta clase es permitir la interacción del especialista con el sistema gestionando las peticiones que le incumben en la propia vista o enviándolas al modelo a través del controlador. Además controla el flujo de eventos desde el controlador hacia la vista de los plugins.

3.5 Diagramas de Secuencia

Los diagramas de secuencia capturan el comportamiento de los casos de uso. Son utilizados para representar las características dinámicas del sistema. Fueron utilizados para especificar, visualizar y documentar el flujo de control de los casos de uso críticos del sistema. Estos diagramas pueden encontrarse en el Anexo 7 del presente documento.

Conclusiones

En este capítulo se realizó la descripción de diseño del sistema, a partir de las funcionalidades propuestas. Se hizo una descripción de los patrones de diseño utilizados, así como los diagramas de paquetes y de clases para un mejor entendimiento del comportamiento del sistema.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

Introducción.

En el presente capítulo se mostrará el diagrama de componentes del sistema y la descripción de cada uno de los componentes que lo integran. Además se hará alusión de los algoritmos utilizados y se detallará la solución que el equipo de desarrollo dio. Y por último se hará un análisis de cada subsistema en cuestión, en el cual se realizarán una serie de pruebas que contribuyen a la calidad del sistema en general.

4.1 Diagrama de componentes.

El diagrama de componentes describe los elementos físicos del sistema y sus relaciones. Para el caso de este trabajo se utilizan los paquetes de clases a nivel lógico descomponiendo de una forma más original el sistema a implementar.

A continuación se muestra el diagrama de componentes realizado para el sistema.

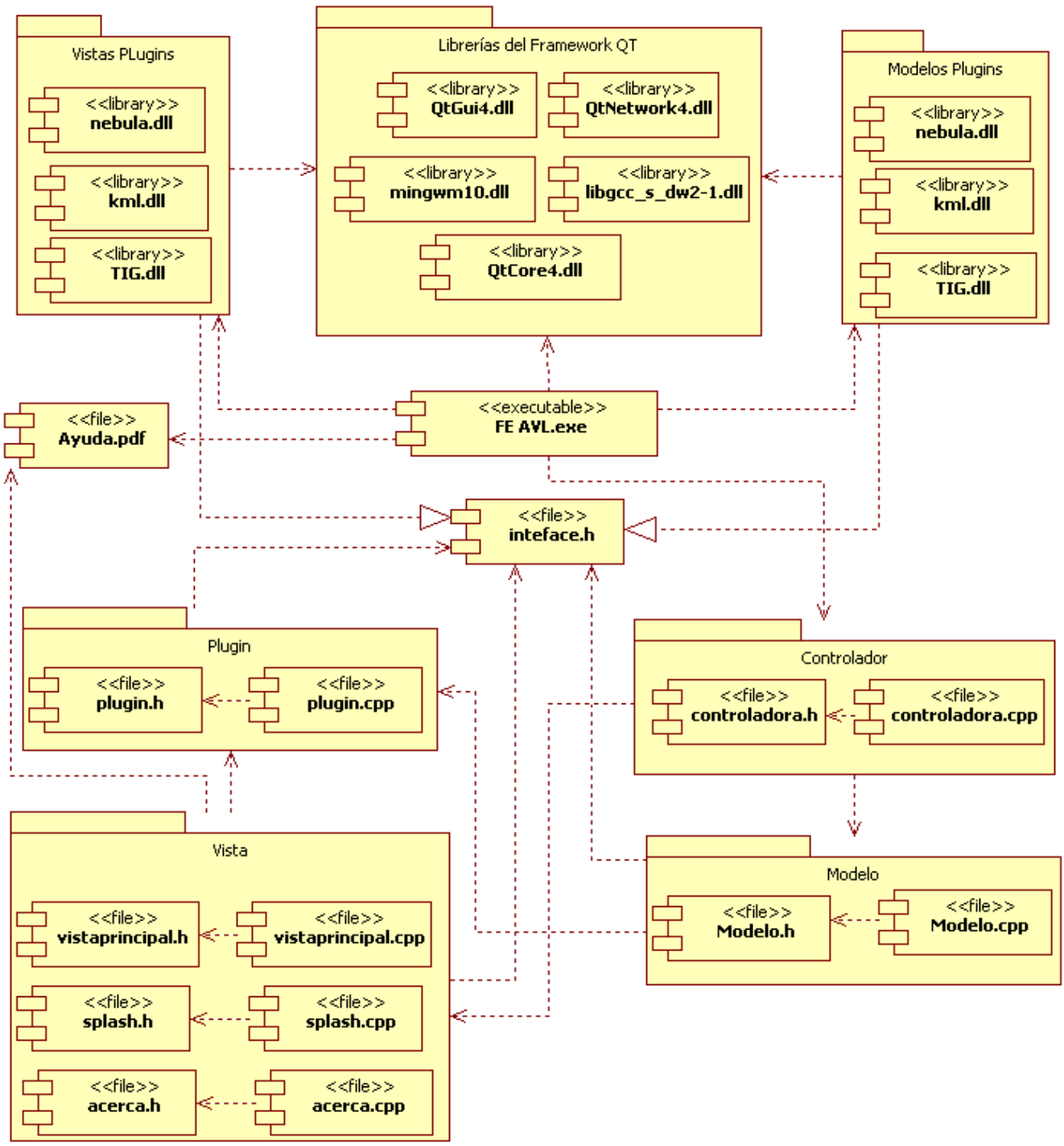

Figura 24: Diagrama de componentes.

4.1.1 Descripción de Componentes.

Paquete de componentes “Vistas Plugins”.

El paquete de componentes “Vistas Plugins” está compuesto por las librerías que forman las vistas de los plugins en el sistema.

Paquete de componentes “Modelos Plugins”.

El paquete de componentes “Modelos Plugins” está compuesto por las librerías que forman los modelos de los plugins en el sistema.

Paquete de componentes “Librerías del Framework QT”.

El paquete de componentes “Librerías del Framework QT” está compuesto por las librerías del Framework QT que se usan en el sistema.

Paquete de componentes “Plugin”.

El paquete de componentes “Plugin” está compuesto por los componentes plugin.h y plugin.cpp, que se encargan de cargar todas las librerías que estén disponibles para el sistema.

Paquete de componentes “Controlador”.

El paquete de componentes “Controlador” está compuesto por los componentes controlador.h y controlador.cpp, que se encargan de controlar todas las peticiones entre las vistas y los modelos.

Paquete de componentes “Vista”.

El paquete de componentes “Vista” está compuesto por los componentes que constituyen controles de usuario creados por los desarrolladores.

Paquete de componentes “Modelo”.

El paquete de componentes “Modelo” está compuesto por los componentes modelo.h y modelo.cpp que controlan y administran las acciones de todos los sub-modelos del sistema.

Ejecutable “FE AVL.exe”.

El ejecutable “FE AVL.exe” es empleado para levantar una instancia del sistema.

Documento “Ayuda.pdf”.

El documento “Ayuda.pdf” es empleado para brindar información acerca del manejo del sistema.

Componente “Interface.h”.

El componente “Interface.h” es empleado como un lenguaje común entre los plugins y el sistema para que se pueda establecer una comunicación.

4.2 Diagrama de Despliegue.

Los Diagramas de Despliegue muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, que generalmente tiene algo de memoria y, a menudo, capacidad de procesamiento. Los nodos se utilizan para modelar la topología del hardware sobre el que se ejecuta el sistema. Los nodos representan el despliegue físico de los componentes. La relación entre un nodo y el componente que despliega puede mostrarse con una relación de dependencia, o listando los nodos desplegados en un compartimiento adicional dentro del nodo.

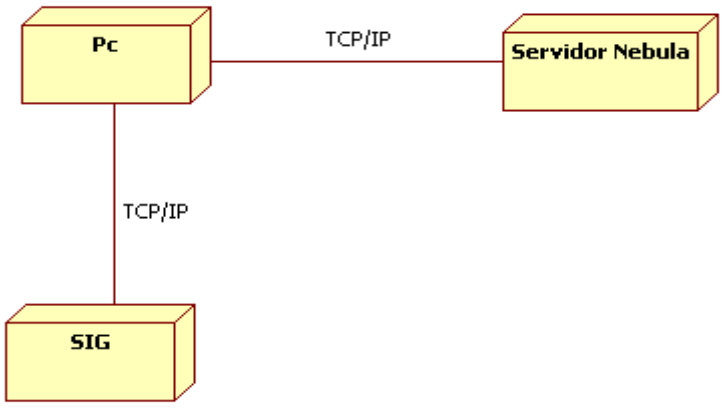

Figura 25: Diagrama de despliegue.

PC

Es una máquina de escritorio o portátil que contiene el Sistema “FE AVL”. Los requerimientos mínimos que debe poseer esta PC son: memoria RAM 256 o superior, sistema operativo Windows XP o superior y Micro ≥ 1.6 GHz.

Servidor NEBULA

Es un servidor que posee varios módulos, entre ellos el servidor AVL NEBULA que se encarga de gestionar el posicionamiento de los dispositivos móviles y portátiles y enviarla a través de TCP/IP.

SIG

Es una máquina de escritorio o portátil que contiene un SIG, este accede por TCP/IP a la información brindada por el sistema “FE AVL” y luego la representa en un sistema de mapas.

4.3 Convenciones de archivos y paquetes.

El proyecto que se crea en Qt Creator se denomina “FE AVL”. Todas las clases y ficheros del sistema están agrupados de acuerdo a la estructura del lenguaje C++ y el framework Qt.

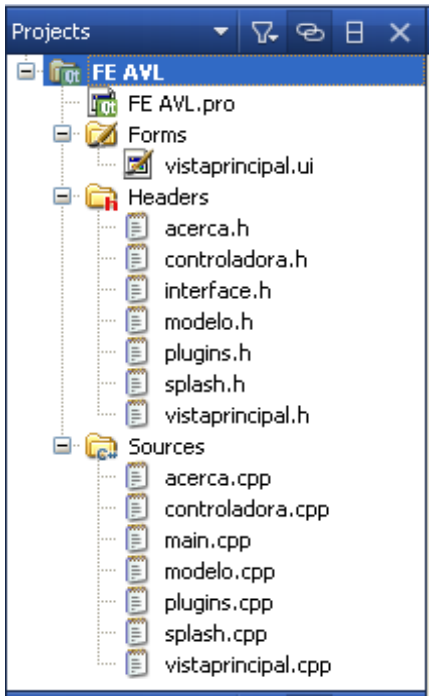

Figura 26: Organización de paquetes de FE AVL

La estructura es la siguiente:

- ✓ **FE AVL.pro** (Archivo que contiene la información necesaria para la compilación del proyecto)
- ✓ **Forms** (Contiene los formularios del Sistema)
- ✓ **Headers (**Contiene las declaraciones de las clases**)**
- ✓ **Sources** (Contiene las implementaciones de de las clases)

4.4 Técnicas de programación.

Para que este sistema cumpla con los requerimientos no funcionales, el equipo de desarrollo se dio a la tarea de buscar técnicas que permitiesen la optimización de consumo del CPU, y que pudiese emplearse con el lenguaje de programación C++. Para esto se escogió el uso de la programación orientada a eventos, donde el framework Qt añade esta característica al lenguaje C++ a través de macros, permitiendo que el sistema no tenga que estar constantemente buscando la ocurrencia de cambios, sino que sean los propios cambios quienes le avisen al sistema. Otro de los aspectos que se estudió fue la gran cantidad de información que cambia constantemente, arrojando que se usaría la estructura de datos QMap porque inserta y elimina en (log(n)) haciendo un uso eficiente de la memoria. También se tuvo en cuenta la gran cantidad de servidores y tareas que el sistema puede tener en ciertas circunstancias por lo que se diseñó un mecanismo para cargar solamente las librerías dinámicas que hagan falta en ese momento.

4.5 Pruebas.

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. La prueba de software es un elemento crítico para la garantía de la calidad del software y están enfocadas principalmente en la evaluación y determinación de la calidad del producto.

4.5.1 Herramientas de Pruebas

Para asegurarse de la calidad del producto se usaron y desarrollaron una serie de herramientas que permiten comprobar el correcto funcionamiento de de los componentes del sistema.

Cliente TIG

Es una aplicación creada por el equipo de desarrollo para comprobar el correcto funcionamiento del plugin TIG.

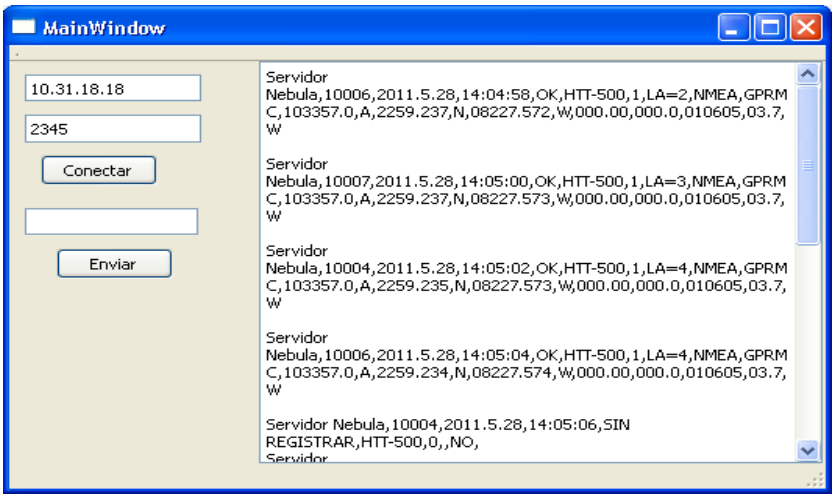

Figura 27: Cliente TIG

NEBULA AVL Server

Es un simulador del Servidor AVL NEBULA creado por el equipo de desarrollo para comprobar el correcto funcionamiento del plugin Servidor NEBULA.

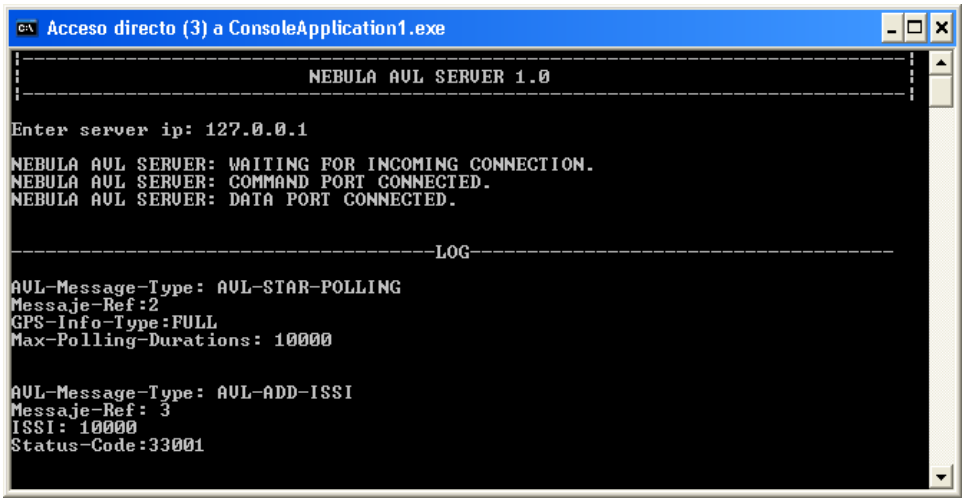

Figura 28: NEBULA AVL Server

Google Earth

Este SIG se utilizó para comprobar que el plugin KML exportara la información en tiempo real y de forma correcta.

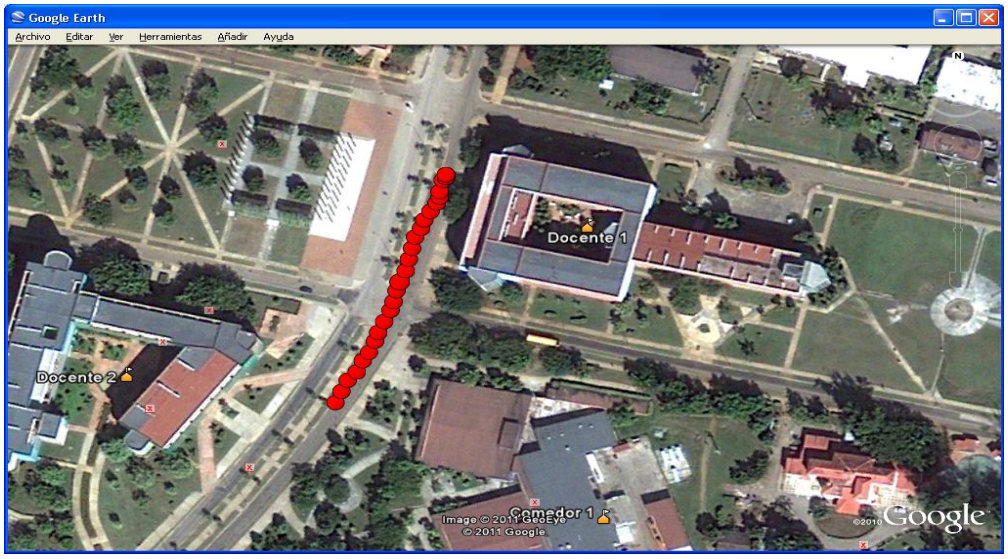

Figura 29: Google Earth

4.5.2 Casos de Prueba

El diseño de casos de pruebas usando el método de caja negra, se utiliza para verificar que la aplicación o el sistema se comportan según los requerimientos establecidos por el cliente, para esto se necesita la lista de los requisitos funcionales del sistema y su descripción. Los casos de pruebas diseñados se encuentran en el Anexo 8 del presente documento.

Conclusiones

En este capítulo se ha podido ver el funcionamiento del sistema a implementar y las técnicas usadas para mejorar su eficiencia. Además se detallaron los diagramas de componentes y de despliegue, y por último se validaron las funcionalidades implementadas, a través de una serie de herramientas y pruebas de caja negra.

CONCLUSIONES GENERALES.

Con el presente trabajo de diploma se presenta el sistema “FRONT END AVL”, el cual permite que la información proveniente del servidor AVL NEBULA pueda ser utilizada por otros SIG. Por tanto se solucionó satisfactoriamente la problemática que existía antes de la construcción del sistema, ya que no se podían usar otros SIG con el servidor AVL NEBULA. De esta forma se le da cumplimiento al objetivo general de la investigación presentada.

Para la construcción del sistema se realizó un estudio del estado de arte sobre los sistemas AVL y los SIG que fue elemental para la realización del mismo. En la implementación se utilizaron herramientas libres dentro de las que se encuentran StarUML la cual permitió realizar el modelado, y como IDE de desarrollo Qt Creator vital para la creación del sistema. Además se usó C++ como lenguaje de programación con el framework Qt y FDD como metodología de desarrollo, partiendo de las peculiaridades del sistema a implementar.

Por lo anteriormente planteado se concluye que los objetivos propuestos para el presente proyecto han sido cumplidos satisfactoriamente, incluyéndose una serie de recomendaciones que se deben tener en cuenta para el trabajo futuro.

RECOMENDACIONES.

- Se recomienda crear una versión del sistema dividida en dos subsistemas: un servicio y un administrador, que permitirá lograr una mayor eficiencia y realizar un mejor manejo del mismo.

- Se recomienda adicionar nuevos plugins al sistema que permitan la integración con los SIG desarrollados en la universidad.

- Se recomienda integrarle un sistema de autenticación para mejorar la seguridad.

REFERENCIAS BIBLIOGRÁFICAS.

1. **Sánchez, Leosdany y Arce, Alberto. 2010.** *El Estandar Tetra.* 2010.
2. ¿Qué es un Socket? - *Definición de Socket* [En línea] [Citado el: 12 de mayo del 2011.] (http://www.masadelante.com/faqs/socket)
3. Infraestructura de Datos Espaciales [En línea] [Citado el: 12 de mayo del 2011.] (http://estadistica.tucuman.gov.ar/idetucuman/sig.php)
4. Tutorial de KML [En línea] [Citado el: 12 de mayo del 2011.] (http://code.google.com/intl/es/apis/kml/documentation/kml_tut.html)
5. Soluciones Avanzadas de Radiocomunicaciones Profesionales [En línea] [Citado el: 1 de junio del 2011.] (http:// www.teltronic.es/file.aspx?Fil_ID=318)
6. Motorola MotoLocator Brochure [En línea] [Citado el: 12 de mayo del 2011.] (http://motorola-latinamerica.hosted.jivesoftware.com/servlet/JiveServlet/download/1587-1-2553/Motorola_MotoLocator_Brochure_ES_121010.pdf)
7. Introducción a Qt [En línea] [Citado el: 12 de mayo del 2011.] (http://www.zonaqt.com/tutoriales/tutorial-de-qt-4-por-zona-qt-0)
8. El lenguaje c++ [En línea] [Citado el: 12 de mayo del 2011.] (http://www.articulandia.com/premium/article.php/12-03-2007El-Lenguaje-C.htm)
9. Metodologías Agiles [En línea] [Citado el: 12 de mayo del 2011.] (http://www.seccperu.org/files/Metodologias%20Agiles.pdf)
10. Lenguaje de Modelado Unificado [En línea] [Citado el: 12 de mayo del 2011.] (http://profejavaoramas.blogspot.com/2010/09/lenguaje-unificado-de-modelado-uml.html)
11. Start UML [En línea] [Citado el: 13 de mayo del 2011.] (http://black-byte.com/review/staruml/)
12. IDE Qt Creator-Completo entorno de desarrollo multiplataforma [En línea] [Citado el: 13 de mayo del 2011.] (http://pixelcoblog.com/qt-creator-completo-entorno-de-desarrollo-multiplataforma/)
13. Spain MVC [En línea] [Citado el: 18 de mayo del 2011.] (http://www.spainmvc.com/)
14. Spain MVC [En línea] [Citado el: 18 de mayo del 2011.] (http://msdn.microsoft.com/es-mx/library/bb972258.aspx#M11)

15. Chain of Responsibility (patrón de diseño) [En línea] [Citado el: 18 de mayo del 2011.] (http://www.diclib.com/cgi-bin/d1.cgi?l=en&base=es_wiki_10&page=showid&id=52014)
16. Patrones Diseño [En línea] [Citado el: 31 de mayo del 2011.] (http://hdlorean.wikidot.com/doc:patrones-diseno)
17. Patrones de diseño y arquitectura [En línea] [Citado el: 31 de mayo del 2011.] (http://www.ecured.cu/index.php/Patrones_de_dise%C3%B1o_y_arquitectura)
18. Alta Cohesión y bajo Acoplamiento. [En línea] [Citado el: 31 de mayo del 2011.] (http://shuster.cs.buap.mx/blog/alta-cohesion-y-bajo-acoplamiento/)

BIBLIOGRAFÍA.

Agile Development. [En línea] [Citado el: 2 de Junio del 2011.] http://www.rspa.com/reflib/AgileDevelopment.html#FDD.

Blanchette, Jasmin y Summerfield, Mark. 2006. *C++ GUI programming with Qt 4.* 2006.

Deparatamento de Ciencias de la Computación. *Tutorial de UML.* [En línea] [Citado el: 2 de Junio del 2011.] http://www.dcc.uchile.cl/~psalinas/uml/introduccion.html.

Ezust, Alan y Ezust, Paul. 2007. *An Introduction to Design.* 2007.

Google. Google Earth. *Tutoriales para principiantes.* [En línea] [Citado el: 2 de junio del 2011.] http://www.google.es/intl/es_es/earth/learn/beginner.html.

Nokia. Qt. *Online Reference Documentation.* [En línea] [Citado el: 2 de Junio del 2011.] http://doc.qt.nokia.com/.

Programacion en Castellano. *Introducción a UML.* [En línea] [Citado el: 2 de Junio del 2011.] http://www.programacion.com/articulo/introduccion_a_uml_181.

Teltronic. CeCoCo Series: Soluciones para Centros de Coordinación y Control. [En línea] [Citado el: 2 de Junio del 2011.] http://teltronic.es/es/PRODUCTOS/CEOCOCO.

—. NEBULA: Infraestructura TETRA. [En línea] [Citado el: 2 de Junio del 2011.] http://teltronic.es/es/PRODUCTOS/NEBULA.

vBulletin. Qt Centre. *About QMacStyle.* [En línea] [Citado el: 2 de Junio del 2011.] http://www.qtcentre.org/threads/16328-About-QMacStyle.

WoltLab. Qt Forum. *Deploying Qt Program.* [En línea] [Citado el: 2 de Junio del 2011.] http://www.qtforum.org/article/35952/deploying-qt-program.html.

GLOSARIO DE TÉRMINOS.

ASCII: -*American Standard Code for Information Interchange / Código Estadounidense Estándar para el Intercambio de Información*- código de caracteres basado en el alfabeto Latino.

Broadcast: Transmisión de un paquete que será recibido por todos los dispositivos en una red.

Ethernet/IP: Protocolo de red en niveles para aplicaciones de automatización industrial.

Front End: Es la parte del software que interactúa con los usuarios.

FTP:-*File Transfer Protocol / Protocolo de Transferencia de Archivos*- es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP.

Gopher: Es un servicio de Internet consistente en el acceso a la información a través de menús.

GUI: -*Graphical User Interface / Interfaz Gráfica de Usuario*- es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

IDE: -*Integrated Development Environment / Entrono de Desarrollo Integrado*- es un programa informático compuesto por un conjunto de herramientas de programación.

IP: El número que identifica a cada dispositivo dentro de una red.

LIP*: -Location Information Protocol / Protocolo de Información de Localización* – es un formato para codificar la información geográfica.

NMEA*:* Es un formato para codificar la información geográfica.

XML: -*Extensible Markup Language / Lenguaje de Marcas Extensible*- es un metalenguaje extensible de etiquetas.

WWW: Es un sistema de distribución de información basado en hipertexto, enlazado y accesible a través de Internet.