Universidad de las Ciencias Informáticas

Facultad 2

Título: FE AVL Versión 2.0, Módulo Reportes.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas.

Autor: Miguel Augusto Gómez Fernández.

Tutores: Ing. Bárbara Triana Morales.
Ing. Alberto Arce Martínez.

19 de Junio de 2012.

AGRADECIMIENTOS.

"Casi cinco años han pasado desde que comencé mis estudios en esta Universidad. Durante todo el camino me han apoyado muchos y muchos han sido los sacrificios realizados para alcanzar esta importante meta. Los resultados de estos años se los debo a todas esas personas que participaron de una forma activa de mi vida y que en todo momento me alentaron para cruzar por encima de las dificultadas y seguir por el sendero que conduce hasta el añorado diploma. Por eso quiero en este momento agradecer:

A mis padres Carmen y Augusto, por su constante aliento y su apoyo económico y moral para que pudiera continuar estudiando.

A mi esposa Rosana y mi hijo Miguel Alejandro, por su apoyo incondicional en los momentos más complicados donde se debía poner en la balanza, por una parte la continuidad de la carrera y por otra el inicio de una vida laboral que nos permitiera el sustento económico.

A mi tía querida, Aida, por ser lo más grande de mi vida y porque sin ella posiblemente no hubiese llegado nunca este día.

A mis suegros Reynaldo y Ester porque desde el primer momento cuando las cosas se pusieron más difíciles, me dijeron: "tienes que terminar".

A mis hermanos no consanguíneos Reinier y Yadel por estar siempre allí en los buenos y malos momentos.

Y a mis compañeros en la Universidad, con los cuales compartí instantes que nunca olvidaré ni aunque pasen cinco años, ni diez, ni aunque nada hubiera sido real y todo volviera a empezar. "

Miguel Augusto Gómez Fernández.

DEDICATORIA.

"A mi familia, inspiración de todas mis victorias y consuelo de todas mis derrotas."

Miguel Augusto Gómez Fernández.

RESUMEN.

En el Centro de Telemática de la Universidad de las Ciencias Informáticas se desarrolló el sistema Front End AVL (FE AVL). El sistema pertenece a la familia de sistema de gestión y control de flotas, área funcional de los sistemas inteligentes de transporte (ITS). Los sistemas de gestión y control de flotas tienen como función principal el monitoreo y control de flotas de dispositivos móviles. Estos sistemas cuentan por lo regular con funcionalidades relacionadas con la elaboración de informes y reportes de tipo históricos con el objetivo de dar soporte a la toma de decisiones y así reducir gastos y mejorar el desempeño de los negocios.

Actualmente el sistema FE AVL no cuenta con un módulo que sea capaz de permitir la elaboración de informes y la generación de reportes de tipo histórico. Debido a esta problemática el Centro de Telemática de la Universidad de las Ciencias Informáticas propone el desarrollo de un módulo de reportes que sea capaz de integrarse con el sistema FE AVL.

En el presente documento se especifican los pormenores del diseño y la implementación del módulo de reportes.

Palabras Clave: *gestión de flotas, ITS, FE AVL, reportes.*

ÍNDICE

INTRODUCCIÓN

Debido a las restricciones económicas a las que se ha visto sujeta nuestro país por el azote del bloqueo económico impuesto por el gobierno de los EE.UU, y teniendo en cuenta que el mundo se encuentra de frente a una nueva era tecnológica: La era de las Tecnologías de la Informática y las Comunicaciones, Cuba constantemente ha buscado alternativas que le permitan avanzar en el campo tecnológico. Ejemplo de lo anteriormente dicho lo constituye La Universidad de las Ciencias Informáticas, fundada en el año 2002.

La Universidad de las Ciencias Informáticas, por sus siglas: UCI, es una institución educacional atípica con respecto al resto de las instituciones educacionales de Cuba. Como todas las universidades del país tiene como misión fundamental la formación de profesionales calificados, en este caso, en la rama de la informática, pero a su vez, cuenta con diversos perfiles de desarrollo dedicados a contribuir económicamente a los sectores más importantes del país. En este sentido, la Universidad se encuentra dividida por centros de desarrollo asociados a un perfil tecnológico determinado.

El Centro de Telemática desarrolla proyectos para empresas nacionales como La Empresa de Telecomunicaciones de Cuba S.A.- ETECSA - e internacionales como la empresa de radiocomunicaciones española TELTRONIC. En dicho centro se desarrolló el Sistema Front End AVL -Automatic Vehicle Location / Localización automática de Vehículo- versión 1.0 (FE AVL), el cual tiene como principal función el monitoreo y control de flotas de dispositivos móviles. Los sistemas de gestión y control de flotas son soluciones basadas en redes de telecomunicaciones ampliamente utilizadas a nivel mundial y con resultados excelentes en la gestión y control de recursos disponibles. Una de las empresas en el ámbito nacional que ha manifestado su interés en este tipo de sistemas es la empresa COPEXTEL S.A.

Los sistemas de gestión y control de flotas brindan grandes facilidades para el monitoreo y supervisión de equipos de radiocomunicaciones en tiempo real, no obstante, uno de los elementos más importante de los mismos es poseer un mecanismo que permita la elaboración de reportes, con el objetivo de que los datos almacenados por dichos sistemas puedan ser utilizados para analizar el comportamiento de las flotas y tomar decisiones que se deriven de dichos análisis. Los reportes, para los sistemas de gestión y control de flota, brindan datos relevantes de acuerdo a determinados criterios, como por ejemplo, determinar las unidades que han tenido una violación de la velocidad establecida o determinar qué unidades han pasado por algún punto geográfico (latitud, longitud) especificado.

Actualmente el sistema FE AVL no posee un sistema de reportes que permita obtener los datos necesarios para la toma de decisiones. Dada la situación problemática anterior se genera el siguiente **Problema a Resolver:** ¿Cómo obtener y visualizar la información útil del sistema de control y gestión de flotas FE AVL para la toma de decisiones?

El **Objeto de Estudio** del presente trabajo se centra en los tipos de reportes realizados por los sistemas de gestión y control de flota.

El **Campo de Acción** queda enmarcado específicamente en el almacenamiento histórico de los datos y la presentación de los mismos en el sistema FE AVL.

Como **Objetivo General** se persigue desarrollar un sistema informático que permita el procesamiento, visualización y generación de reportes de los datos contenidos en el sistema FE AVL.

Consecuentemente se definieron los siguientes objetivos específicos:

- Diseñar e implementar el Almacén de Datos del sistema.
- Crear reportes.
- Generar reportes en formato PDF.
- Generar reportes en formato HTML.
- Imprimir reportes.

Para lograr un desarrollo satisfactorio y darle seguimiento a los objetivos trazados, se plantearon las siguientes preguntas de la investigación:

- ¿Existen sistemas para la gestión de flotas que puedan ser utilizados en la toma de decisiones?
- ¿Qué beneficios traería la generación de reportes para estos sistemas?
- ¿Qué aportes pudiera ofrecer la solución planteada?
- ¿Cuál es la metodología de desarrollo adecuada para guiar el proceso de desarrollo?
- ¿Qué lenguaje de modelado se adecua mejor a la metodología escogida?
- ¿Cuáles son las diferentes tecnologías y los diferentes formatos con los cuales trabajará el sistema?
- ¿Qué herramientas serán necesaria para diseñar e implementar la solución?

Para dar cumplimiento al objetivo general propuesto, se desarrollan las siguientes tareas de la investigación:

- Estudio del modelo dimensional y los almacenes de datos.
- Identificación de las tecnologías y herramientas necesarias para desarrollar el sistema.

- Estudio del arte de los diferentes tipos de reportes que generan los sistemas de gestión de flotas.
- Descripción del Sistema.
- Análisis de la arquitectura a utilizar.
- Diseño e implementación del Almacén de Datos del sistema.
- Implementación de la aplicación.
- Diseño e implementación de los casos de prueba.
- Realización de las pruebas al sistema.

Para cumplir con el objetivo propuesto se aplican los métodos de investigación analítico – sintético para realizar el estudio del arte de los diferentes tipos de reportes que generan los sistemas de gestión de flotas, y para realizar un estudio de las metodologías de desarrollo, tecnologías y herramientas de desarrollo, simplificando y organizando el análisis de todos los datos recopilados. El método histórico – lógico para realizar un estudio de los antecedentes y tendencias actuales de los mecanismos de reportes de los sistemas de gestión de flotas, para estudiar los antecedentes y tendencias en el uso de las metodologías de desarrollo, tecnologías y herramientas de desarrollo. Y el método experimental para la realización de pruebas al sistema que verifican el correcto funcionamiento de las funcionalidades implementadas.

Con propósito organizativo y estructural, se ha decidido dividir el presente trabajo en 5 capítulos:

Capítulo 1: Se realiza un estudio del arte de los reportes que generan los sistemas de gestión de flotas. Se detallan las herramientas y tecnologías que se utilizarán en el desarrollo del sistema así como los lenguajes y metodologías de desarrollo.

Capítulo 2: Describe la situación problemática y los procesos que serán objeto de automatización. Aborda aspectos esenciales a tener en cuenta del dominio y los requisitos.

Capitulo 3: Describe el proceso de diseño e implementación del almacén de datos que dará soportes a los históricos y demás datos de interés para el análisis y la toma de decisiones.

Capítulo 4: Se realizan los diagramas de paquetes del diseño y las tarjetas CRC – Class, Responsibility, Colaboration / Clases, Responsabilidades y Colaboración - del sistema. Se definen los patrones de arquitectura y de diseño a utilizar.

Capítulo 5: Se confecciona el diagrama de despliegue, el diagrama de componentes, se describen las pruebas realizadas al sistema y los resultados de las mismas.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

Introducción

En el presente capítulo se exponen los Sistemas Inteligentes de Transporte (ITS) y los Sistemas de Gestión y Control de Flotas como un área funcional de los primeros. Se expone el estado del arte de los mecanismos de reportes de los sistemas de gestión y control de flotas. Por otra parte, se trata la metodología de desarrollo de software y el lenguaje de modelado que se utilizará en el ciclo de desarrollo de la solución informática. Se realiza una descripción de cada una de las tecnologías y herramientas utilizadas para dar cumplimiento al objetivo general propuesto en la investigación y especialmente se abordan los elementos fundamentales referentes a los sistemas OLAP – *On Line Analytical Process / Proceso Analítico en Línea* -.

1.1 Sistemas Inteligentes de Transporte (ITS)

Los ITS o Sistemas Inteligentes de Transporte abarcan desde la primera central automatizada de control de semáforos hasta los actuales sistemas capaces de determinar alcances entre vehículos, y frenar o acelerar el automóvil en el que están instalados. [20]

La base de todo ITS está en la recogida de datos, integrándolos de una forma comprensible para el ser humano y suministrándole la información de forma elaborada y comprensible. [20]

Esta información debe recibirse a ser posible en tiempo real, por lo que la comunicación debe realizarse tanto por medio de cableado entre estaciones fijas a un Centro de Control, como por medios inalámbricos a estaciones móviles (vehículos). [20]

Un colectivo particularmente interesado en las tecnologías ITS es el de la gestión de flotas de transporte, tanto de pasajeros como de mercancías. Estos sistemas permiten conocer en todo momento la situación de toda la flota hasta monitorizar los parámetros de conducción de sus vehículos. Para el transporte de pasajeros, el enlace entre varios medios (compañías de autobuses, trenes de cercanías y de largo recorrido, ferris...) otorga una flexibilidad añadida a las redes de transporte público, cuyo aumento de usuarios beneficia inmediatamente tanto al medio ambiente como a la descongestión de las carreteras. [20]

1.2 Sistemas de Gestión y Control de Flotas

La Gestión de Flotas no es más que el control sobre los recursos de transporte disponibles por parte de una organización. Puede incluir una variedad de funcionalidades como:

- Seguimiento vehicular.

- Organización de unidades en flotas.
- Control sobre el kilometraje recorrido por parte de un vehículo.
- Control del consumo de combustible.
- Control de los conductores al frente de cada vehículo.
- Control de recorridos y rutas.
- Control de áreas de trabajo.
- Control de las fechas de vencimiento de las revisiones técnicas de los vehículos.
- Obtención de informes de diversos tipos como soporte a la toma de decisiones.

Actualmente existen muchos sistemas software que se dedican a cumplir estas funciones. Los sistemas de control y gestión de flotas pueden clasificarse a grandes rasgos en tres campos: el transporte de pasajeros (urbano o interurbano, con varias compañías e incluso modos de transporte que se integran en un único sistema), la gestión de vehículos de emergencia, y la gestión de flotas de transporte privadas. Todos coinciden en la optimización de los recursos de que disponen, pero sus necesidades y complejidad son muy diferentes. Son absolutamente necesarios para optimizar las flotas de vehículos de emergencia, y sus ventajas en el transporte intermodal urbano son rápidamente apreciadas allí donde se están implantando. En cuanto a las flotas de compañías privadas de transporte, su coste y la gran atomización del sector hacen que su implantación sea más lenta. [20]

1.3 Estado del Arte de los mecanismos de reportes de los sistemas de gestión de flotas.

1.3.1 Software de Localización Global AVL.

Global AVL permite la generación de los siguientes tipos de reportes: [1]

- Control de velocidad.
- Consumo de combustible.
- Datos de Can Bus -Controller Area Network-.
- Informes de tacógrafo.
- Informes de los mensajes enviados a los conductores.

Global AVL no permite la generación de reportes de localización, ni de alarmas. Tampoco permite la integración de nuevos reportes al sistema lo cual implica rigidez en cuanto al acceso a los datos relevantes por parte de los usuarios para la toma de decisiones. Depende de modelos de localizadores GPS de vehículos específicos reduciendo así las posibilidades de comunicación con

otros sistemas que poseen módulos GPS, como por ejemplo Terminales que utilizan el estándar de radiocomunicaciones TETRA. El sistema está soportado sobre una plataforma eminentemente privativa como es ASP .NET v3.5.

1.3.2 FlotasNet de Fagor Electrónica. Plataforma Internet.

FlotasNet permite la realización de una amplia gama de informes entre los que destacan: [2]

- Informe de Actividad de Vehículo.
- Informe de Velocidad: Control de la velocidad del vehículo.
- Informe de % Velocidad Vehículo: Control estadístico de los modos de conducción de la flota.
- Informes de Conductor.
- Informes de Trayectos.
- Informes de Comunicaciones.
- Informes de Mensajería, Alarma, y Eventos.

No obstante la amplia gama de reportes que permite generar FlotasNet, no permite la integración de nuevos reportes al sistema lo cual implica que de cambiar las necesidades de información de los usuarios el sistema podría dejar de ser del todo útil. FlotasNet también requiere de conexión constante e ininterrumpida a Internet y es una solución altamente costosa basada en tecnologías privativas.

1.3.3 FlotasMap de Fagor Electrónica. Plataforma Intranet.

FlotasMap es la versión Intranet del sistema FlotasNet. Permite realizar los mismos tipos de reportes de FlotasNet y tiene básicamente sus mismas limitaciones, además de utilizar herramientas de tipo propietaria para la realización de sus funciones, y de ser un sistema que posee tipo de licencia privativa. [3]

1.3.5 Soluciones de Monitoreo y Control de Flota de MICRONAV

Entre los tipos de reportes que se pueden generar con las soluciones de MICRONAV destacan: [4]

- Seguimiento en tiempo diferido (históricos) vehículo a vehículo o de toda la flota en ventana independiente.
- Informe de actividad (recorridos parciales y totales, tiempos de conducción y descanso, velocidades máximas, direcciones postales de los apagados, entre otros).
  - Informe gráfico de velocidades.
  - Informe gráfico de tiempos de conducción y descanso.

- Informe de posiciones minuto a minuto o de 15 en 15 segundos.
- Informe de verificación de cumplimiento de itinerarios.
- Informe de eventos.
- Informe tipo tacógrafo.

- Exportación de cualquier informe a formato PDF o EXCEL y almacenamiento en formato de la aplicación (.hst).

Los sistemas de Micronav no permiten la integración de nuevos reportes al sistema lo cual implica que de cambiar las necesidades de información de los usuarios el sistema podría dejar de ser del todo útil. Micronav presenta limitaciones en cuanto a la comunicación con dispositivos que no utilizan el estándar GSM. Utiliza como cartografía Google Maps, dependiendo totalmente de este servicio para su funcionamiento.

1.3.6 Soluciones de Monitoreo y Control de Flotas en Cuba

Cuba tiene poco desarrollo en la gestión y control de flotas, no obstante se han desarrollado algunos sistemas de gestión y control de flotas, tal es el caso de MOVIL WEB. A continuación se presenta una breve descripción de esta aplicación:

Aplicación Web para el control de vehículos, tanto en tiempo real como en diferido, basada en los servicios de mapas disponibles en la Infraestructura de Datos Espaciales de la República de Cuba. Esta herramienta permite el monitoreo de móviles de manera remota sobre una red de comunicaciones, posibilitando reconstruir el comportamiento del móvil en un determinado periodo de tiempo, reconstruyendo su trayectoria y analizando su velocidad, detenciones, salidas fuera de la ruta planificada, etc. a través de la información almacenada en la base de datos histórica. Dicha herramienta se viene aplicando desde Abril del 2006 y hasta el momento se han logrado reducciones de hasta un 30% del combustible asignado a los vehículos, además de un mayor control de los recorridos realizados y datos sobre los tiempos de carga y descarga que permiten a las bases de transporte realizar una mejor evaluación de su eficiencia. La centralización de la aplicación y las bases de datos posibilita el monitoreo, la auditoría y control, por las entidades autorizadas, sobre el trabajo de cada base de trasporte. [21]

En la práctica se ha constatado que MovilWeb no ofrece un efectivo control en tiempo real de los vehículos y además no puede ser integrada con otros tipos de infraestructuras como la infraestructura TETRA. Tampoco permite la reutilización de sus mecanismos para la generación de reportes por lo que no es viable su uso por parte del sistema FE AVL.

En la Universidad de las Ciencias Informáticas se desarrollan también algunas soluciones sobre este tema. El ejemplo más visible con el que se cuenta es el sistema FE AVL desarrollado en el Centro de Telemática de dicha Universidad el cual hasta la versión 1.0 permite la gestión y configuración de servidores *AVL- Automatic Vehicle Location / Rastreo Automatizado de Vehículos* - obteniendo la información de los *terminales* que se encuentren conectados a esta infraestructura y guardándola en formato KML - *Keyhole Markup Language* - para posteriormente poder representar dicha información en un SIG – *Sistema de Información Geográfica* – como, por ejemplo, Google Earth. El sistema FE AVL no permite la generación de reportes de ningún tipo.

1.3.7 Inconvenientes para la utilización en el sistema FE AVL de los mecanismos de reportes existentes en el mercado para la gestión y control de flotas

Luego de haber conocido algunos de los más utilizados sistemas de gestión y control de flotas y los principales tipos de reportes que estos pueden generar, se ha arribado a la conclusión de que no se adecuan a las necesidades y por tanto no brindan una solución efectiva al problema planteado. La principal razón de la inviabilidad de estos sistemas se debe a que la solución deseada debe cumplir en su totalidad con las siguientes restricciones:

- El sistema debe de ser capaz de elaborar todos los reportes que sean solicitados por el cliente, permitiendo la inclusión de nuevos reportes.
- El sistema de reportes debe integrarse como un módulo del sistema FE AVL.
- De utilizarse un sistema de reportes externo, este debe ser libre y gratuito y debe cumplir con los dos puntos antes descritos.
- El sistema debe cumplir todas las prestaciones definidas por el cliente.

1.4 Metodologías de desarrollo y lenguaje de modelado.

Las metodologías de desarrollo de software definen un marco de trabajo que nos permiten organizar, controlar y gestionar el proyecto. Actualmente predominan dos tipos de metodologías: Las metodologías “ágiles” o “ligeras” y las metodologías “pesadas” o “robustas”. Una metodología de desarrollo de software usualmente utiliza un lenguaje de modelado para comunicar sus procesos y documentar de una manera gráfica los proyectos. Por otra parte, existen metodologías específicas para el desarrollo de data warehouses y data marts las cuales tienen como función principal guiar el proceso de desarrollo de almacenes de datos.

A continuación se describen las metodologías usadas en el proceso de desarrollo de software en el diseño y la implementación del módulo de reportes y del almacén de datos así como el lenguaje de modelado usado conjuntamente con la metodología de desarrollo de software.

1.4.1 Scrum.

Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos. [7]

La elección de Scrum como metodología de desarrollo de software está fundamentada por la flexibilidad que brinda el proceso ante la poca definición de los requisitos del sistema y la urgencia de realizar entregas que aportasen nuevas funcionalidades en periodos de tiempo breves.

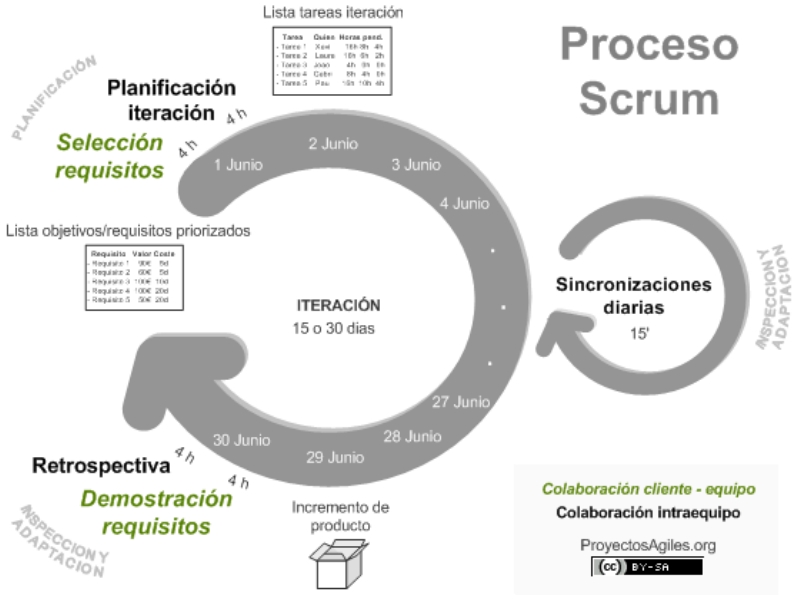


Figura 1: Proceso Scrum. [7]

1.4.2 Hefesto v2.0

HEFESTO es una metodología propia, cuya propuesta está fundamentada en una muy amplia investigación, comparación de metodologías existentes, experiencias propias en procesos de confección de almacenes de datos. Cabe destacar que HEFESTO está en continua evolución, y se han tenido en cuenta, como gran valor agregado, todos los feedbacks que han aportado quienes han utilizado esta metodología en diversos países y con diversos fines. [22]

HEFESTO resulta ideal para el desarrollo del almacén de datos del sistema debido a su especial facilidad de integración con metodologías de desarrollo de software ágiles, por tomar lo mejor de las metodologías de desarrollo de data warehouse, como la metodología de Kimbal, sin dejar de ser fácil de aplicar y por su capacidad de obtener buenos resultados en un breve periodo de tiempo.

1.4.3 UML (Unified Modeling Language / Lenguaje Unificado de Modelado) v2.0.

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. UML incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. Está pensado para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código así como generadores de informes. La especificación de UML no define un proceso estándar pero está pensado para ser útil en un proceso de desarrollo iterativo. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos. [8]

1.5 Lenguajes y Tecnologías Utilizadas.

1.5.1 QT Framework v4.7.0.

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. [5]

Qt extiende el lenguaje C++, a través de macros y meta información, mientras se mantiene apegado a él. Entre las características que agrega Qt a C++ se encuentran: los Bucle foreach, y el manejo de señales y slots. [5]

Las principales razones por las cuales se eligió el famework Qt para el desarrollo de la solución propuesta son las facilidades que ofrece para el trabajo con plugins mediante la librería QtPlugins y para el manejo de eventos. Otra de las razones de la elección es la existencia de precedentes

con buenas referencias en la utilización del framework ya que el sistema FE AVL está implementado utilizando dicho framework.

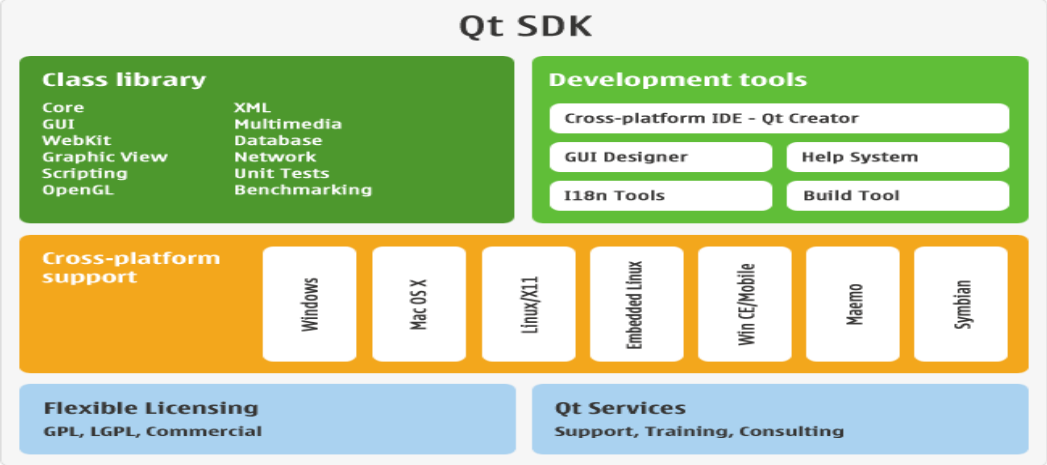


Figura 2: Framework Qt.

1.5.2 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. [6]

La decisión de utilizar C++ como lenguaje de programación para el desarrollo de la solución propuesta está marcado por la elección de Qt como framework de desarrollo debido a que dicho lenguaje es el que mejor se integra al mencionado framework.

1.5.3 Almacenes de Datos. Modelo Dimensional.

Un Almacén de datos (Bodega de Datos, Data warehouse) es una integrada colección de datos que contiene datos procedentes de sistemas del planeamiento del recurso de la empresa (*SAP*, *CRM*, *ERP*) y de otros sistemas relacionados al negocio. Los almacenes de datos contienen información que se subdivide a veces en unidades lógicas más pequeñas, llamadas Data Marts. [16]

El almacén se nutre de estas fuentes de datos externas (Sistemas *OLTP*), las cuales se utilizan para el trabajo diario, quedando disponible para que las aplicaciones puedan consultarlo. Luego, estas pueden presentar los datos de diferentes formas y en diversos formatos.

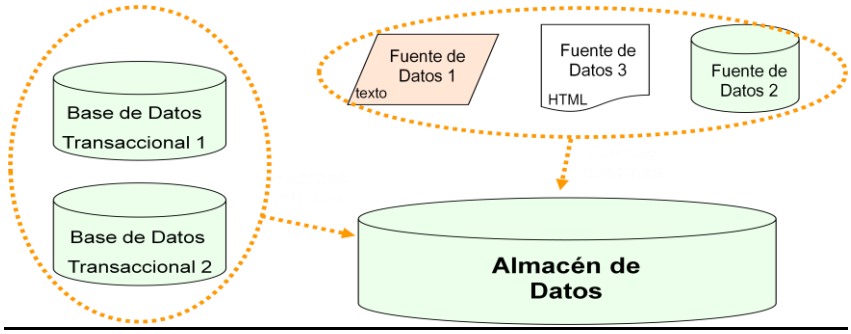


Figura 3: Almacenes de datos.

Estos tres conceptos: sistemas *OLTP*, *Data warehouse* y *Data Mart* conforman la arquitectura básica de los almacenes de datos. De este modo, se identifican dos diseños principales en esta arquitectura:

- **Arquitectura de Kimbal:** Propone la creación de *Data Marts* a partir de diferentes sistemas *OLTP*, luego los *Data Marts* darán origen al *Data warehouse*. Esta arquitectura se caracteriza por ser relativamente sencilla de utilizar y muy estable en presencia de cambios.

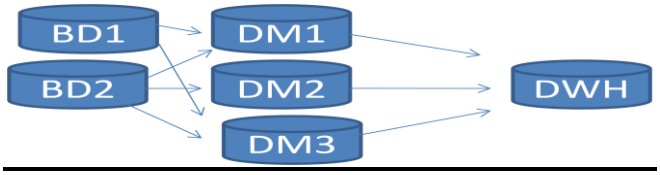


Figura 4: Arquitectura propuesta por Ralph Kimbal.

- **Arquitectura de Inmon:** Propone la creación del *Data warehouse* a partir de diferentes sistemas operacionales, luego este dará origen a diferentes *Data Marts*. Propone el *modelo relacional* como base de su diseño, lo que la hace más flexible que la de Kimbal, aunque consume mucho más tiempo de desarrollo y es más propensa al fracaso respecto a esta.

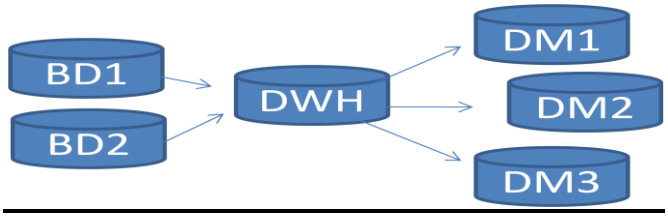


Figura 5: Arquitectura propuesta por William H. Inmon.

Tanto Kimbal como Inmon coinciden que el mejor *modelo de datos* que se puede usar para la implementación de los *Data Marts* es el *modelo dimensional* pero Inmon es partícipe de usar el *modelo relacional* para la creación del *Data warehouse*.

1.6 Herramientas Utilizadas

1.6.1 StarUML v5.0.2.1570.

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. [10]

Se selecciona StartUml como herramienta de modelado ya que se adecua perfectamente a las necesidades de desarrollo permitiendo la elaboración de cualquier diagrama definido en el estándar UML lo que la pone al mismo nivel que Visual Paradigm en cuanto a las funcionalidades requeridas y además es libre.

1.6.2 Qt Creator v2.0.1.

Qt Creator es un entorno de desarrollo (IDE) multiplataforma muy completo. [11]

Principales características de Qt Creator: [11]

- 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 integrado.
- 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. [11]

QtCreator se selecciona debido a ser el IDE por excelencia para el desarrollo de software utilizando el framework Qt.

1.6.3 PostgresSQL v8.2

PostgreSQL es un sistema de gestión de base de datos relacional orientada a objetos y libre, publicado bajo la licencia BSD.

Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyada por organizaciones comerciales. Dicha comunidad es denominada el PGDG (*PostgreSQL Global Development Group*). [23]

Algunas de sus principales características son, entre otras: [23]

- Alta concurrencia.
- Amplia variedad de tipos nativos.
- Soporta llaves foráneas (foreign keys).
- Soporta disparadores (triggers).
- Soporta Vistas.
- Integridad transaccional.
- Herencia de tablas.
- Tipos de datos y operaciones geométricas.

Se elige PostgresSQL como Sistema Gestor de Base de Datos (SGBD) debido a su característica de alta concurrencia, por ser un SGBD extremadamente seguro, por ser un SGBD libre y por soportar la implementación de la tecnología de data warehousing.

1.6.4 Pentaho Data Integration v4.2.1

El Pentaho Data Integration es uno de los componentes del proyecto Pentaho (también conocido como Kettle). Se trata de un motor de integración de datos al que se puede acceder utilizando una interfaz gráfica para definir trabajos y transformaciones. El Kettle soporta la ejecución de los trabajos en equipos convencionales (PC), pero también cuenta con funciones para el procesamiento distribuido: clusters de computadoras y computación en nube.

Definiéndolo en términos menos formales, con el Pentaho Data Integration se puede de manera muy simple tomar datos de una fuente (archivos locales y remotos, bases de datos, repositorios…), aplicar un procesamiento a dichos datos (filtros, condiciones, cálculos, consultas), y almacenar los resultados en un destino (archivos, base de datos, repositorio…). [24]

Figura 6: Funcionamiento de Pentaho Data Integration. [24]

Su elección está dada por la facilidad de construcción de los procesos ETL –Extraction Transformation and Load / Extracción Transformación y Carga – mediante la herramienta Spoon perteneciente a esta suite la cual permite realizar el diseño de estos procesos de forma gráfica y por pertenecer a la familia del software de libre.

Conclusiones

En el presente capítulo se ha realizado un estudio acerca de los Sistemas Inteligentes de Transporte y dentro de estos los Sistemas de Gestión de Flotas. Se ha hecho un estudio de los tipos de reportes generados por las principales aplicaciones de gestión y control de flota que existen actualmente en el mercado, exponiendo las razones de por qué no es viable su empleo en la resolución del problema planteado. Se ha dado un breve preámbulo de Scrum como metodología de desarrollo de software a utilizar y de Hefesto como la metodología de desarrollo a utilizar en el diseño del almacén de datos, así como de UML como lenguaje de modelado. Por otra parte se han conocido las diversas tecnologías y herramientas que se utilizarán en el desarrollo del sistema.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

Introducción

En el presente capítulo se hace la descripción de la solución al problema planteado. Se detalla la estructura del sistema, así como los principales elementos conceptuales que lo componen. También se hace la descripción de las funcionalidades del mismo así como un listado de los requisitos no funcionales. Además se verán las historias de usuario asociadas a cada una de estas funcionalidades y la planificación por iteración de las tareas de desarrollo.

2.1. Objeto de automatización

Actualmente, en la versión 1.0, el sistema FE AVL no cuenta con un módulo para generar reportes basados en datos históricos. Para dar solución a la problemática anterior se hace necesaria, en primera instancia, la creación de un almacén de datos que contendrá los datos históricos del sistema FE AVL necesarios para la elaboración de los reportes. Posteriormente se desarrolla un sistema cuyo propósito general es establecer conexión con el almacén de datos y proporcionar al usuario del sistema FE AVL un mecanismo para la elaboración de diferentes tipos de reportes que se visualizarán en formato tabular. El sistema debe permitir además, exportar los reportes a formato pdf y html e imprimirlos.

2.2. Descripción de la solución

El módulo de reportes es un *plugin* (extensión) que se integra al sistema FE AVL en su versión 2.0. A su vez cada uno de los reportes que se generan son *plugins* (extensiones) que se integran al módulo de reportes brindando una gran escalabilidad, permitiéndole crecer en funcionalidades además de ser tolerante a futuros cambios en la estructura del almacén de datos. Cada uno de los reportes tiene la responsabilidad de establecer conexión con el almacén de datos, utilizando los parámetros de conexión brindados por el sistema FE AVL, y extraer del almacén de datos la información necesaria para la elaboración de los reportes. Una vez extraída la información, se crea un objeto de reporte que puede ser interpretado por el núcleo del sistema de reportes, permitiéndole visualizar la información en formato tabular. Los datos mostrados pueden ser estructurados para dar formato al documento y podrá exportarse el documento a formato pdf, html, así como imprimirlo.

2.3. Estructura del sistema

El sistema de reportes está compuesto fundamentalmente por: el almacén de datos y el módulo de reportes.

El almacén de datos almacena toda la información necesaria para realizar reportes de tipo históricos en forma de tablas. La fuente de datos de la cual se nutre el DWH es la base de datos OLTP utilizada por el sistema FE AVL para sus transacciones diarias en la gestión y control de flotas, causa que justifica la creación del almacén de datos, utilizando este último solamente para la elaboración de los reportes históricos contribuyendo así a un mejor rendimiento del sistema en general. La información relevante para el propósito del sistema de reportes se extrae del mencionado OLTP, se transforma haciendo tareas de limpieza y transformación de datos, y se carga al almacén de datos de manera regular mediante la herramienta ETL *Spoon*, perteneciente a la suite de *Pentaho Data Integration*.

Por su parte, el módulo de reportes mantiene la misma arquitectura del sistema FE AVL. Se trata de una estructura flexible que permite que el sistema vaya incrementando a medida de las necesidades de los usuarios la cantidad y los tipos de reportes que estos pueden hacer. 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, el núcleo busca en tiempo de ejecución en el directorio *Plugin* del módulo de reportes, verificando que cada *plugin* esté correctamente implementado para poder adicionarlos y de esta forma extender la cantidad y los tipos de reportes que se pueden realizar.

Cada reporte es un *plugin*. Los reportes varían de un tipo a otro en cuanto a los datos de entradas necesarios para la realización de las consultas al almacén de datos y los datos de salida que conforman el reporte, pero todos están compuestos por una clase de interfaz cuya función es interactuar con el usuario y recibir los parámetros de entrada del reporte, y un conjunto de clases de lógica de negocio encargadas de establecer la conexión con el almacén de datos, enviar las consultas pertinentes y estructurar los datos resultantes en una entidad que posteriormente será interpretada por el núcleo del sistema para la visualización del reporte. Cada plugin está compuesto por dos librerías dinámicas, una para el modelo (clases de la lógica de negocio) y otra para la vista (clase de interfaz), en este caso la función de controlador la cumple el núcleo del módulo de reportes el cual recibe de la interfaz un mensaje con los parámetros de entrada del reporte, los pasa al modelo y finalmente este elabora el reporte enviándolo nuevamente al núcleo para su presentación. De esta manera se implementa el patrón de arquitectura MVC – *Model View Controller / Modelo Vista Controlador* -.

Entre las ventajas que ofrece la estructura propuesta se encuentra la promoción a la escalabilidad del sistema debido a la utilización de reportes como plugins, esto permite que siempre se puedan agregar nuevos reportes al sistema. Además, la independencia de los reportes con respecto al núcleo ofrece cierta flexibilidad en caso de que cambiase la estructura del almacén de datos ya que se tendrían que reprogramar nuevamente los reportes pero no afectaría al núcleo del sistema. Por otra parte se pueden desarrollar un gran número de reportes en relativamente poco tiempo ya que al ser independientes, los reportes, unos de otros, el equipo de desarrollo puede trabajar paralelamente en varios. En el momento de desarrollar pruebas al sistema, las mismas pueden hacerse de manera independiente pues no se necesita probar todo el sistema sino solamente el plugin de reporte que se está desarrollando, teniendo en cuenta las pruebas de integración con el núcleo del sistema. Otra de las ventajas destacables es con respecto a la actualización y el soporte pues no se necesita compilar todo el sistema sino solamente la parte que se desea actualizar o corregir. Por otra parte, en el proceso de comercialización se le brinda al cliente la posibilidad de elegir los reportes que desea en su sistema.

2.4 Modelo de dominio

Producto de que la solución propuesta no forma parte de un proyecto llave en mano con algún cliente externo, sino que tiene como cliente al Centro de Telemática de la Universidad de las Ciencias Informáticas, no existe un negocio bien definido. Por cuanto se hizo imposible la realización de un modelo de negocio, y como alternativa para mejor comprensión del sistema se decidió realizar un modelo de dominio. Seguidamente se presenta el modelo de dominio realizado donde se evidencian los principales conceptos identificados en la solución del problema planteado. Además se hace una descripción textual de estos conceptos y sus relaciones.

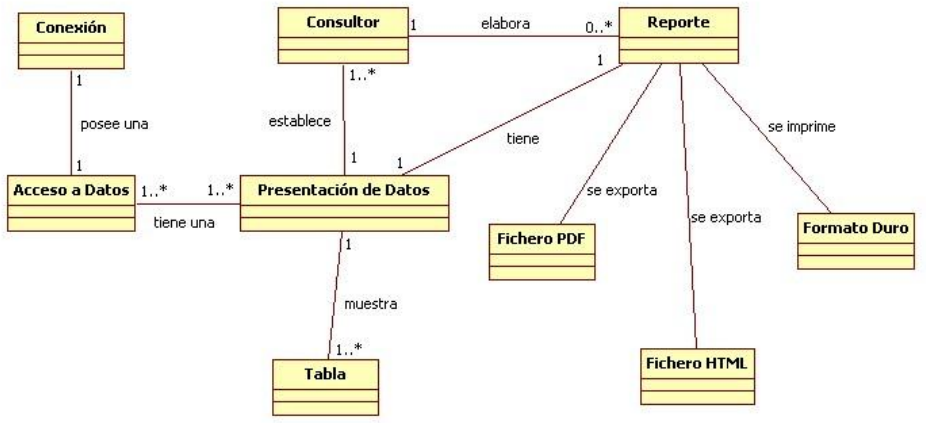


Figura 7: Modelo de dominio.

- **Consultor:** Representa al usuario que utiliza el sistema. Este *elabora* los reportes y *establece* el tipo de presentación que desea darle. Puede además exportar estos reportes a formato pdf y html así como imprimirlo.
- **Presentación de Datos:** La presentación de los datos es elección del especialista el cual la configura. Los datos se presentan en formato tabular. Cada reporte tiene una presentación de los datos.
- **Tabla:** Formato mediante el cual se presenta la información.
- **Reporte:** Documento que *elabora* el consultor, contiene datos relevantes para la toma de decisiones y *tiene* una presentación particular de los datos. Puede ser exportado a fichero pdf o html. También se pueden imprimir.
- **Fichero PDF:** Representa un fichero PDF.
- **Fichero HTML:** Representa un fichero HTML.
- **Formato Duro:** Representa un reporte *impreso*.
- **Acceso a Datos:** Mecanismo mediante el cual se accede a los datos guardados en el almacén de datos y se recuperan para su posterior presentación.
- **Conexión:** Representa el vínculo entre el almacén de datos y la aplicación. Una conexión con el almacén de datos es abierta por el mecanismo de acceso a datos.

El proceso comienza cuando el consultor decide elaborar un reporte, introduce los datos de entrada del reporte y ejecuta el reporte, posteriormente el mecanismo de acceso a datos asociado al reporte abre una conexión con el almacén de datos y envía las consultas, luego recibe los datos y crea el

reporte que queda listo para su presentación en formato tabular. Posteriormente el consultor puede exportar el reporte a un fichero pdf o html o llevarlo a formato duro mediante su impresión.

2.5 Funcionalidades del sistema

2.5.1 Requisitos funcionales

Los requisitos funcionales son las capacidades de funcionamiento con las que debe cumplir un software y están estrechamente relacionados con el sentido de ser del mismo. Scrum define como artefacto para enumerar los requisitos funcionales la Pila del Producto. A continuación se muestra la pila del producto del sistema FE AVL, Módulo de Reportes.

ID	EI	C	EA	Funcionalidad
1	15	0.8	20	Realizar reporte.
2	20	0.9	23	Cargar reporte.
3	1	0.1	1	Crear nuevo documento de informe.
4	5	0.3	3	Crear documentos de informes a partir de una plantilla.
5	3	0.2	3	Exportar informe a formato HTML.
6	3	0.2	3	Exportar informe a formato PDF.
7	3	0.2	3	Imprimir informe.
8	4	0.3	3	Pre visualizar informe.
9	4	0.3	3	Editar informe.
10	10	0.6	13	Dar formato a informe.
11	5	0.3	3	Insertar tabla.
12	3	0.3	2	Insertar imagen desde archivo.
13	2	0.3	3	Insertar sección de informe.
14	2	0.3	3	Insertar líneas.
15	8	0.5	8	Configurar formato de las tablas.
16	30	1	24	Integrar módulo a Sistema FE AVL.

∑ EI	∑ C/ 16	∑ EA
118	**0.41**	**118**

ID: Identificador de funcionalidad
EI: Estimación inicial (en días)
C: Complejidad (estimado entre 0 y 1)
EA: Estimación ajustada (en días)

∑ **EI:** Total de la estimación inicial (en días)
∑ **C/16:** Complejidad promedio de la aplicación
∑ **EA:** Total de la estimación ajustada o real (en días)

2.5.2 Requisitos no funcionales

Los requisitos no funcionales son características que poseen los sistemas que no están relacionadas directamente con el sentido de ser del software pero que en muchos casos determinan el éxito de un producto.

Apariencia o interfaz externa

- Interfaz con un diseño sencillo que contenga pocos gráficos, sin dejar de ser amigable, interactiva, fácil de usar y profesional. Se utilizarán colores afines al sistema FE AVL y al Centro de Telemática como el azul, el blanco y el gris.

Hardware.

- Micro ≥ 1.6 GHz
- Memoria RAM ≥ 512 MB
- Impresora.

Rendimiento

- Se debe poder integrar un ilimitado número de plugins de reportes.
- Se debe poder exportar la información a ficheros pdf y html rápidamente.

Soporte

- El sistema contará con un instalador para su 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 poseer un manual de usuario.
- El sistema debe ser fácil de actualizar.

Diseño

- El lenguaje de programación que se usará es C++.
- Para el diseño del sistema debe ser utilizada la metodología Scrum, usando complementariamente el lenguaje de modelación UML v2.0 y como herramienta para llevarlo a cabo el StarUML v5.0.2.1570.
- Para el diseño y soporte del almacén de datos se utilizará la metodología de desarrollo Hefesto v2.0 y el servidor de base de datos PostgresSQL v8.2.
- Para el diseño de las ETL se utilizará Pentaho Data Integration v4.2.1.

Ayuda

- El sistema brindará a los usuarios una ayuda en formato pdf que podrá ser consultada desde el mismo sistema.

Seguridad

- El sistema debe asegurar la integridad y disponibilidad de la información de los reportes garantizando que no existan datos incongruentes en al almacén de datos y que todos los usuarios autorizados puedan acceder a la información.
- Se debe informar al usuario en caso de que el servidor de bases de datos no esté disponible, no exista conexión, o cualquier otro percance relacionado con la conectividad.

2.6 Actores del sistema.

Los actores del sistema son aquellos usuarios que interactúan con el sistema y que realizan una, varias o todas las funcionalidades que brinda este. Según el modelo de dominio y la descripción del sistema se han identificado los siguientes actores del sistema:

Actor	Descripción
Consultor	Realiza todas las funcionalidades enumeradas en la pila del producto del módulo de reportes.
Replicador	Realiza la carga y actualización del almacén de datos.

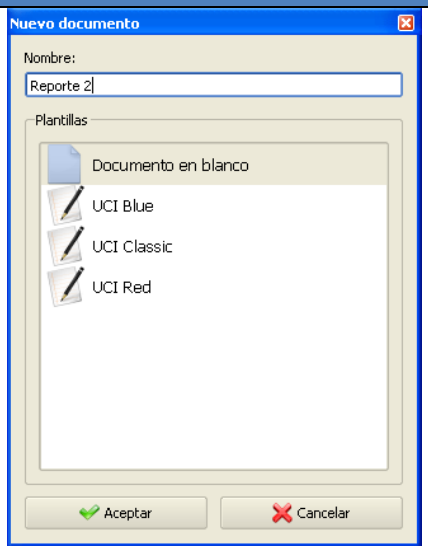
2.7 Historias de usuarios

Las historias de usuario son un artefacto que describen las funcionalidades del sistema desde el punto de vista del product owner (cliente). Para ver íntegramente las historias de usuario relacionadas con las funcionalidades descritas en la pila del producto vea el Anexo 1 de la versión digital de este documento. A continuación se presentan las principales historias de usuario:

ID:	1	**Nombre:**	**Realizar reporte.**		
Como cliente deseo que se puedan realizar reportes de diferentes tipos. Para esto el Sistema deberá consultar al almacén de datos extrayendo cualquier información solicitada. Los reportes deberán presentar: ➢ Título del reporte. ➢ Parámetros con los que fue configurado el reporte. ➢ Datos del reporte. Los datos del reporte pueden estar contenidos en una o varias tabla, en dependencia del tipo de reporte.					
Estimación (días):	15	**Prioridad:**	1	**Dependiente de:**	-
Pruebas de aceptación					

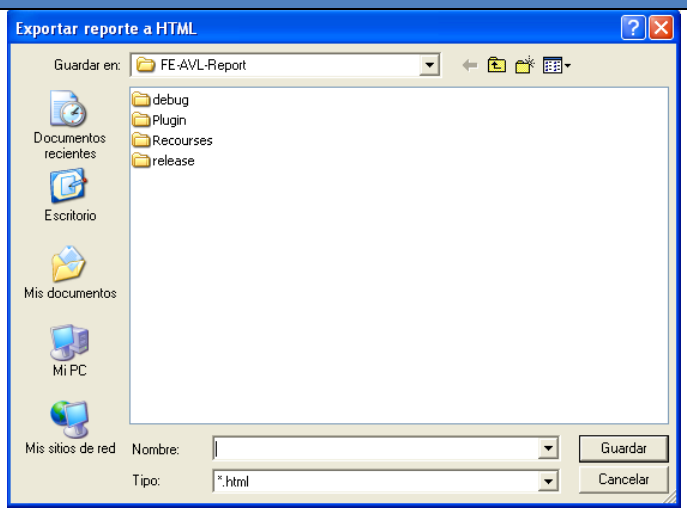
➢ Verificar que los datos del reporte son correctos y que se muestran todos los datos que se mencionaron anteriormente.
Responsable: Miguel Augusto Gómez Fernández.

ID:	3	**Nombre:**	**Crear documento de informe.**		
Como cliente deseo que el sistema permita crear documentos para la elaboración de informes que contengan los reportes deseados. El sistema deberá poder manejar varios documentos al mismo tiempo.					
Estimación (días):	1	**Prioridad:**	4	**Dependiente de:**	-
Pruebas de aceptación					
➢ Verificar que se ha agregado el nuevo documento de informe al sistema.					
Responsable:	Miguel Augusto Gómez Fernández.				
Prototipo de interfaz de usuario					

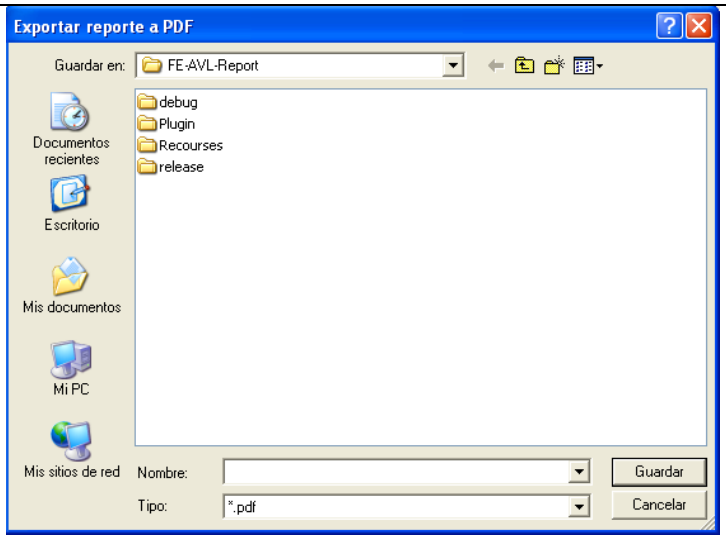


ID:	5	**Nombre:**	**Exportar informe a formato HTML.**		
Como cliente deseo que el sistema permita guardar los informes elaborados en formato html.					
Estimación (días):	3	**Prioridad:**	6	**Dependiente de:**	3
Pruebas de aceptación					

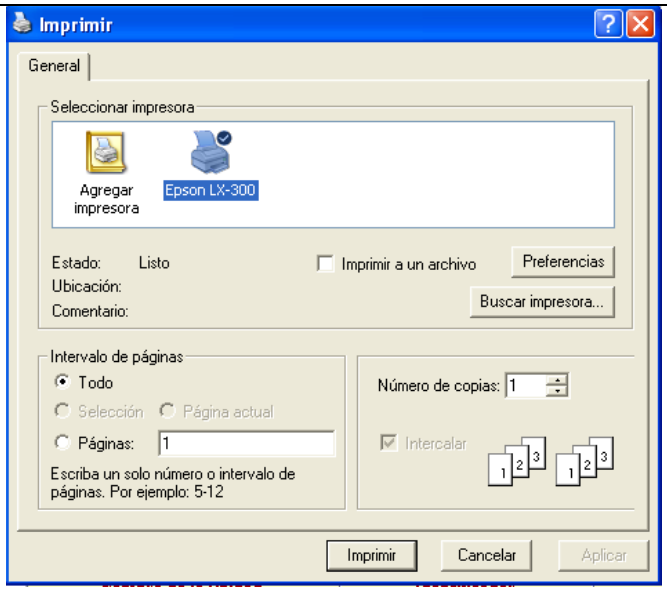
➢ Verificar que se ha guardado el documento en formato HTML.	
Responsable:	Miguel Augusto Gómez Fernández.
Prototipo de interfaz de usuario	



ID:	6	**Nombre:**	**Exportar informe a formato PDF.**		
Como cliente deseo que el sistema permita guardar los informes elaborados en formato PDF.					
Estimación (días):	3	**Prioridad:**	5	**Dependiente de:**	3
Pruebas de aceptación					
➢ Verificar que se ha guardado el documento en formato PDF.					
Responsable:	Miguel Augusto Gómez Fernández.				
Prototipo de interfaz de usuario					



ID:	7	Nombre:	Imprimir informe.		
Como cliente deseo que el sistema permita configurar la impresión e imprimir los informes elaborados.					
Estimación (días):	3	**Prioridad:**	7	**Dependiente de:**	3
Pruebas de aceptación					
➢ Verificar que todas las configuraciones realizadas para la impresión se reflejan en el documento impreso. ➢ Verificar que el sistema realiza la impresión de los documentos de manera adecuada.					
Responsable:	Miguel Augusto Gómez Fernández.				
Prototipo de interfaz de usuario					



2.8 Pila de sprint (Sprint Backlog)

La pila de sprint es un artefacto que lista las funcionalidades que se van a implementar en un sprint (iteración) determinado. Esta contiene tareas de desarrollo requeridas para completar elementos de la pila del producto (product backlog). Para ver la pila de sprint para todos los sprints del proceso de desarrollo vea el Anexo 2 de la versión digital de este documento.

Conclusiones

En el presente capítulo se ha descrito la solución al problema planteado así como los requisitos funcionales y no funcionales del sistema. Además se presentaron y se describieron los principales conceptos y sus relaciones en el modelo de dominio y se hizo la descripción de cada una de las funcionalidades en las historias de usuario del sistema además de identificar en que sprint se van a implementar estas funcionalidades.

CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN DEL ALMACÉN DE DATOS

Introducción

En el presente capítulo se realiza mediante metodología HEFESTO el diseño del almacén de datos que dará soporte a los registros históricos así como a la información relevante para la toma de decisiones de directivos y personal en general que trabaje en el control y la gestión de flotas. También se verá la implementación de los procesos ETL que cargarán inicialmente y mantendrán actualizado el almacén de datos.

3.1. Introducción a la metodología de diseño de Data warehouse HEFESTO.

El diseño y la implementación de un almacén de datos constituyen un proyecto de desarrollo de software en sí. Existen algunos muy buenos materiales y libros que describen este proceso. De ahí que algunas de estas técnicas se hayan generalizado convirtiéndose en metodologías de desarrollo que nos guían a través este proceso.

HEFESTO es una de las metodologías para el diseño e implementación de almacenes de datos que se integra muy bien con cualquier ciclo de vida de desarrollo de software y especialmente con metodologías de desarrollo ágiles tales como Xp y Scrum, por cuanto se hace muy propicio su utilización en el desarrollo del data mart propuesto como parte de la solución del sistema FE AVL versión 2.0, módulo de reportes.

A continuación se describen cada uno de los pasos realizados para el desarrollo del Data Mart del módulo de reportes del sistema FE AVL en su versión 2.0.

3.2 Análisis de los Requerimientos

El primer paso en la construcción de un almacén de datos es la identificación de los requerimientos de información de los usuarios de dicho almacén. Esto se hará a través de preguntas realizadas con el fin de vislumbrar los objetivos de la organización y/o el negocio. Luego, estas preguntas serán analizadas y como resultado se obtendrán los indicadores y perspectivas relevantes en la construcción del almacén de datos. Por último se confeccionará el modelo conceptual visualizando así los resultados de este primer paso.

3.2.1 Identificar preguntas

De acuerdo con los resultados obtenidos, derivados de diversas entrevistas y reuniones con el propietario del producto (product owner) y demás interesados, en donde se formularon preguntas tales como:

- ¿Qué tipo de datos se hace necesario guardar para llevar registros históricos que constituyan información relevante para la gestión de flotas?
- ¿Qué datos son relevantes para la toma de decisiones de directivos con respecto a la gestión de flotas?
- ¿Hasta qué punto es necesario mantener almacenada información antigua?
- ¿Cuál es el nivel de granularidad que deben mantener las dimensiones del DWH en comparación con las tablas y atributos correspondientes en el sistema OLTP?

Se derivaron los siguientes requerimientos de información:

RI	Requerimiento de información
1	Se desea llevar un registro histórico de localización de acuerdo a los siguientes factores: Espacio de tiempo en que se localizó la unidad, unidad localizada, información de trama enviada por la unidad, punto geográfico en que se localizó la unidad, grupo al que pertenece la unidad (si acaso pertenece a alguno de los grupos existentes) y flota a la que pertenece dicho grupo y por consiguiente la unidad.
2	Se desea llevar un registro histórico de alarmas activadas de acuerdo a los siguientes factores: Espacio de tiempo en que se activó la alarma, unidad que activó la alarma, punto geográfico en el cual se activó la alarma, grupo en donde se activó la alarma (en caso de que la alarma fuera emitida por una unidad que pertenece a un grupo existente), flota en la cual se activó la alarma, ruta a la que pertenece la alarma (en caso de que la alarma sea de tipo ruta), reja virtual a la que pertenece la alarma (en caso de que la alarma sea de tipo reja) y tipo de alarma activada.
3	Se desea conocer la cantidad de alarmas activadas de acuerdo a cualquier criterio de las perspectivas relacionadas.

3.2.2 Identificar indicadores y perspectivas

Las perspectivas y los indicadores obtenidos de los requerimientos de información identificados son:

RI	Perspectivas	Indicadores

1	➢ Unidad. ➢ Información de unidad. ➢ Punto geográfico. ➢ Tiempo. ➢ Grupo. ➢ Flota.	-
2	➢ Unidad. ➢ Ruta. ➢ Reja virtual. ➢ Tipo de Alarma. ➢ Tiempo. ➢ Punto Geográfico. ➢ Grupo. ➢ Flota.	-
3	➢ Unidad. ➢ Ruta. ➢ Reja virtual. ➢ Tipo de Alarma. ➢ Tiempo. ➢ Punto Geográfico. ➢ Grupo. ➢ Flota.	➢ Cantidad de alarmas.

3.2.3 Modelo Conceptual

A continuación se presenta el modelo conceptual resultante de los datos recopilados en los pasos anteriores:

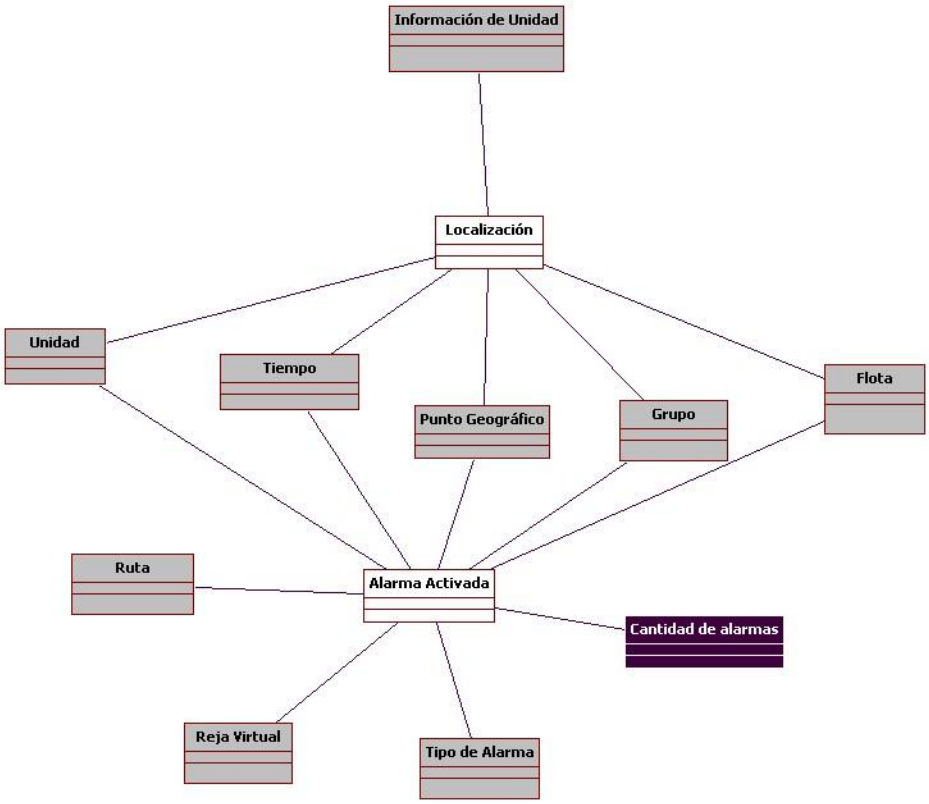


Figura 8: Modelo conceptual del almacén de datos.

Las entidades de color blanco representan las relaciones, las de color gris las perspectivas y las de color negro los indicadores.

3.3 Análisis de los OLTP

A continuación se hace un análisis de las fuentes OLTP existentes para determinar cómo se calcularán los indicadores y para establecer la correspondencia entre el modelo conceptual creado en el paso anterior y las fuentes de datos. Luego se definirán que campos se incluirán en cada perspectiva. Por último se ampliará el modelo conceptual con la información obtenida en este paso.

3.3.1 Conformar indicadores

Los indicadores identificados se calcularán de la siguiente forma:

Indicador	Hechos	Func. Sum	Aclaraciones

Cantidad de alarmas	fact_alarm_triggered	COUNT	Cantidad de alarmas representa la cantidad de alarmas activadas: 1. En una reja en particular. 2. En una ruta en particular. 3. Por una unidad en particular. 4. De un tipo determinado. 5. En un punto geográfico determinado. 6. En un grupo determinado. 7. En una flota determinada. 8. En un periodo de tiempo determinado. 9. Cualquier combinación de los criterios anteriores.

3.3.2 Establecer correspondencias

A continuación se establecerán las correspondencias entre el sistema OLTP del sistema FE AVL y el modelo conceptual del almacén de datos.

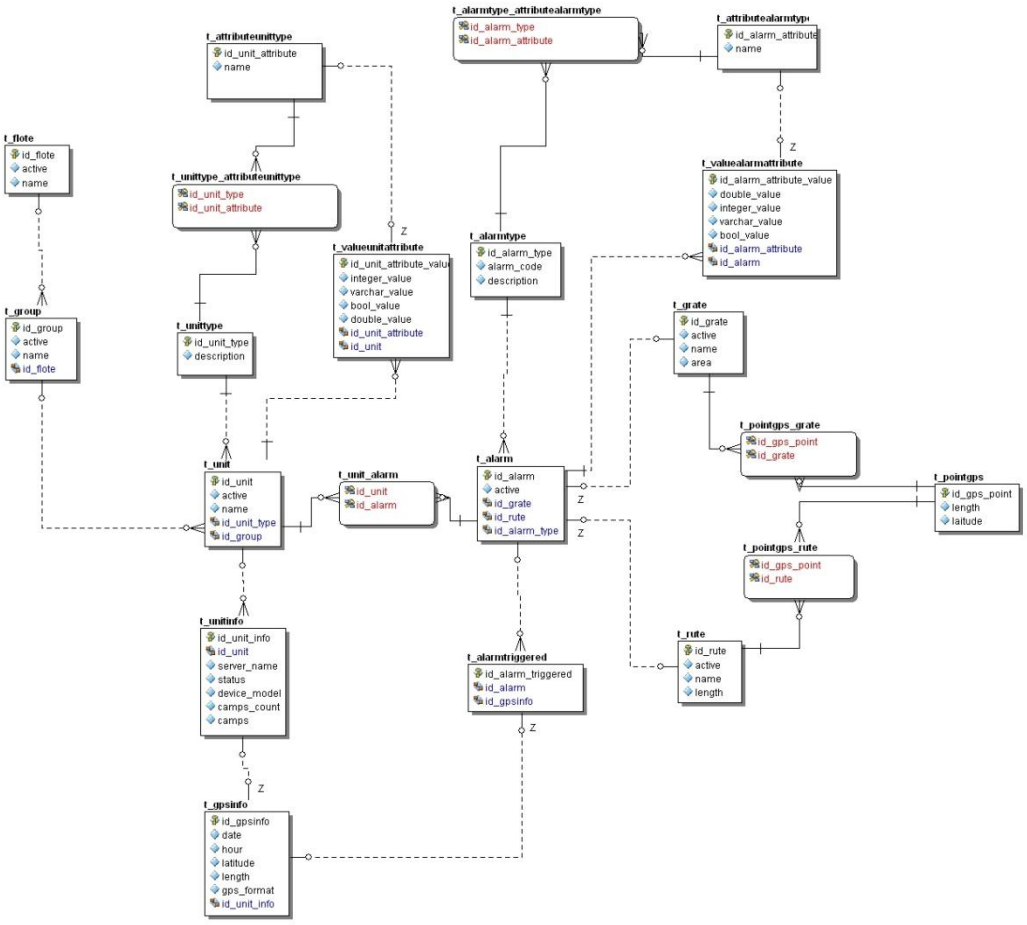


Figura 9: Base de datos OLTP del sistema FE AVL.

Las relaciones entre el sistema OLTP y el modelo conceptual son las siguientes:

- La tabla *t_unit* se relaciona con la perspectiva *unidad*.
- La tabla *t_flote* se relaciona con la perspectiva *flota*.
- La tabla *t_group* se relaciona con la perspectiva *grupo*.
- La tabla *t_unitinfo* se relaciona con la perspectiva *información de unidad*.
- El campo *date* y el campo *hour* de la tabla *t_gpsinfo* se relacionan con la perspectiva *tiempo.*
- El campo *latitude*, *length* y *gps_format* de la tabla *t_gpsinfo* se relacionan con la perspectiva *punto geográfico.*
- La tabla *t_rute* se relaciona con la perspectiva *ruta*.
- La tabla *t_grate* se relaciona con la perspectiva *reja virtual*.
- La tabla *t_alarmtype* se relaciona con la perspectiva *tipo de alarma*.

3.3.3 Nivel de Granularidad

A continuación se definen y se hace una descripción de los campos que contendrá cada perspectiva:

Perspectiva	Campo	Descripción
Información de Unidad (vea el Anexo 6 de la versión digital de este documento)	server_name	Representa el nombre del servidor AVL al que está conectada la unidad.
	status	Representa el estado de la unidad.
	device_model	Representa el modelo de la unidad.
	camps_count	Representa la cantidad de campos adjuntos al final de la trama TIG.
	camps	Representa el conjunto de campos adjuntos al final de la trama TIG.
Unidad	id_unit	Representa el identificador único que posee cada unidad.
	name	Representa un nombre o alias por el que se conoce a la unidad.
Tiempo	date	Representa una fecha en el formato (dd/MM/año).
	time	Representa la hora en el formato (hh:mm:ss).
	day	Representa un día del mes.
	month	Representa un mes del año en formato de texto (enero, febrero, marzo, abril, mayo, junio, julio, agosto, septiembre, octubre, noviembre, diciembre).

	year	Representa un año.
	day of week	Representa el día de la semana (lunes, martes, miércoles, jueves, viernes, sábado, domingo).
	fifteen days	Representa la quincena del mes (primera, segunda).
	semester	Representa un semestre del año (primero, segundo).
	trimester	Representa un trimestre del año (primero, segundo, tercero).
	fourt month period	Representa un cuatrimestre del año (primero, segundo, tercero, cuarto).
Punto geográfico	latitude	Representa la latitud en que se encuentra una unidad.
	lenght	Representa la longitud en que se encuentra una unidad.
	gps_format	Representa el formato GPS en que está enviando la información una unidad.
Grupo	name	Representa el nombre del grupo.
Flota	name	Representa el nombre de la flota.
Ruta	name	Representa el nombre de la ruta.
	length	Representa la longitud de la ruta en kilómetros (km).

Reja Virtual	name	Representa el nombre de la reja virtual.
	area	Representa el área de la reja virtual en (km2).
Tipo de Alarma	alarm_code	Representa el código universal de la alarma.
	description	Es una descripción del funcionamiento de la alarma.

3.3.4 Modelo conceptual ampliado

A continuación se presenta el modelo conceptual con los campos del OLTP que se corresponden con las perspectivas identificadas.

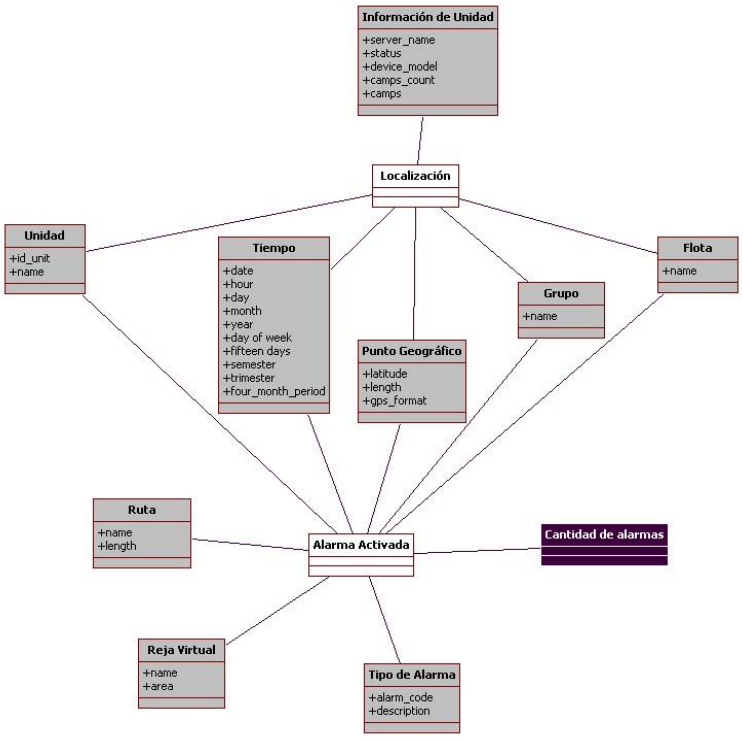


Figura 10: Modelo conceptual ampliado del almacén de datos.

3.4 Modelo lógico del Data warehouse

Luego de haber obtenido el modelo conceptual ampliado corresponde desarrollar el modelo lógico del almacén de datos. Para ello primero se define el tipo de modelo que se utilizará y luego se llevarán a cabo las acciones correspondientes al caso para diseñar las tablas de dimensiones y de hechos. Por último se realizarán las uniones pertinentes entre estas tablas.

3.4.1 Tipo de modelo lógico del Data warehouse

El tipo de modelo lógico que se ha decidido implementar es constelación debido a que se cuenta con más de una relación.

3.4.2 Tablas de dimensiones

A continuación se presentará el diseño de las tablas de dimensiones:

Perspectiva: Información de unidad

- La nueva tabla de dimensión tendrá el nombre de *dim_unitinfo*.
- Se agregará una clave principal con el nombre *id_unit_info.*
- Se agregará un nuevo campo con el nombre *id_oltp* el cual representa el identificador que tenía el registro en el sistema OLTP.

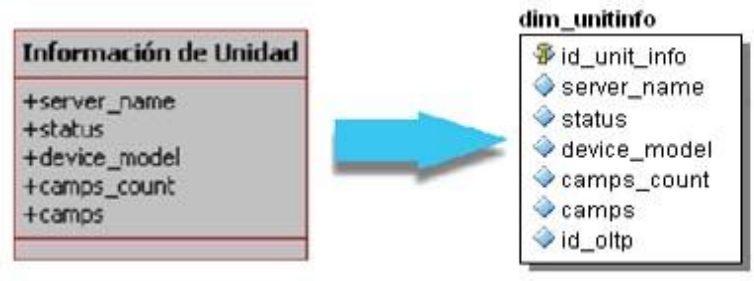


Figura 11: Mapeo de la perspectiva Información de Unidad a la tabla de dimensión dim_unitinfo.

Perspectiva: Unidad

- La nueva tabla de dimensión tendrá el nombre de *dim_unit*.
- Se cambiará el nombre del campo *id_unit* por *identified.*
- Se agregará una clave principal con el nombre *id_unit.*
- Se agregará un nuevo campo con el nombre *id_oltp* el cual representa el identificador que tenía el registro en el sistema OLTP.

Figura 12: Mapeo de la perspectiva Unidad a la tabla de dimensión dim_unit.

Perspectiva: Tiempo

- La nueva tabla de dimensión tendrá el nombre de *dim_time*.
- Se cambiará el nombre del campo *hour* por *time*.
- Se agregará una clave principal con el nombre *id_time.*
- Se agregará un nuevo campo con el nombre *id_oltp* el cual representa el identificador que tenía el registro en el sistema OLTP.

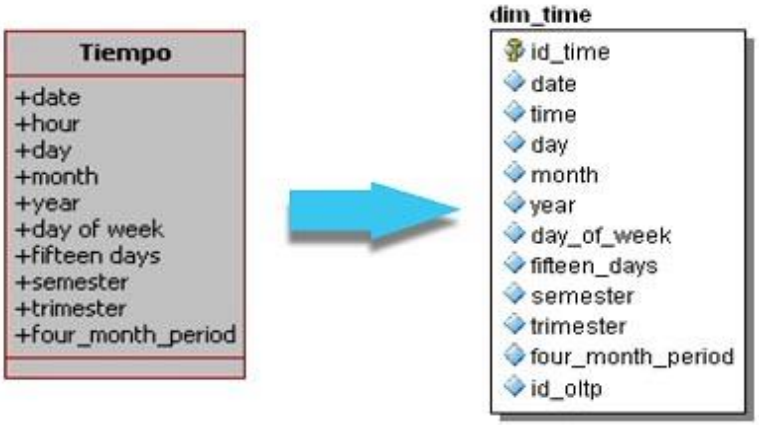


Figura 13: Mapeo de la perspectiva Tiempo a la tabla de dimensión dim_time.

Perspectiva: Punto geográfico

- La nueva tabla de dimensión tendrá el nombre de *dim_geography*.
- Se agregará una clave principal con el nombre *id_geography.*
- Se agregará un nuevo campo con el nombre *id_oltp* el cual representa el identificador que tenía el registro en el sistema OLTP.

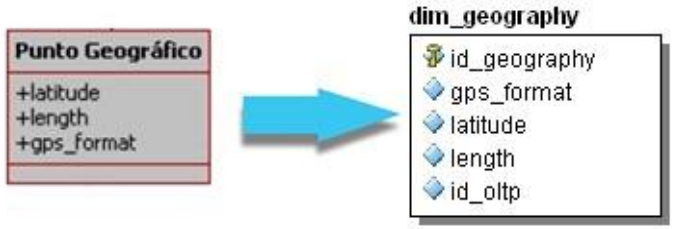


Figura 14: Mapeo de la perspectiva Punto Geográfico a la tabla de dimensión dim_geography.

Perspectiva: Grupo

- La nueva tabla de dimensión tendrá el nombre de *dim_group*.
- Se agregará una clave principal con el nombre *id_group.*
- Se agregará un nuevo campo con el nombre *id_oltp* el cual representa el identificador que tenía el registro en el sistema OLTP.

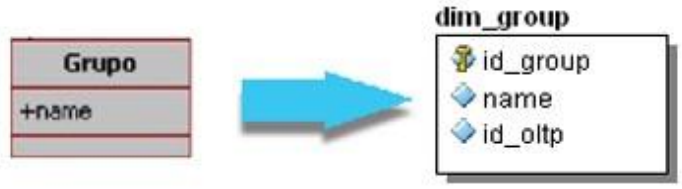


Figura 15: Mapeo de la perspectiva Grupo a la tabla de dimensión dim_group.

Perspectiva: Flota

- La nueva tabla de dimensión tendrá el nombre de *dim_flote*.
- Se agregará una clave principal con el nombre *id_flote.*
- Se agregará un nuevo campo con el nombre *id_oltp* el cual representa el identificador que tenía el registro en el sistema OLTP.

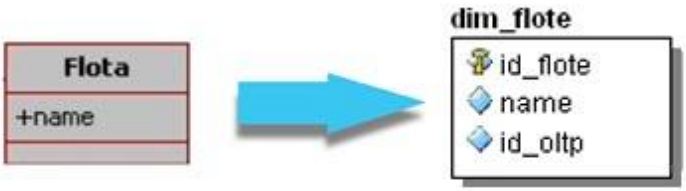


Figura 16: Mapeo de la perspectiva Flota a la tabla de dimensión dim_flote.

Perspectiva: Ruta

- La nueva tabla de dimensión tendrá el nombre de *dim_rute*.
- Se agregará una clave principal con el nombre *id_rute.*

- Se agregará un nuevo campo con el nombre *id_oltp* el cual representa el identificador que tenía el registro en el sistema OLTP.

Figura 17: Mapeo de la perspectiva Ruta a la tabla de dimensión dim_rute.

Perspectiva: Reja virtual

- La nueva tabla de dimensión tendrá el nombre de *dim_grate*.
- Se agregará una clave principal con el nombre *id_grate.*
- Se agregará un nuevo campo con el nombre *id_oltp* el cual representa el identificador que tenía el registro en el sistema OLTP.

Figura 18: Mapeo de la perspectiva Reja Virtual a la tabla de dimensión dim_grate.

Perspectiva: Tipo de alarma

- La nueva tabla de dimensión tendrá el nombre de *dim_alarmtype*.
- Se agregará una clave principal con el nombre *id_alarm_type.*
- Se agregará un nuevo campo con el nombre *id_oltp* el cual representa el identificador que tenía el registro en el sistema OLTP.

Figura 19: Mapeo de la perspectiva Tipo de Alarma a la tabla de dimensión dim_alarmtype.

Debido que se aplicó en dos ocasiones el patrón de de diseño de base de datos Entidad-Atributo-Valor al diseño lógico del almacén de datos, con el objetivo de lograr una flexibilidad con respecto a los atributos que pueden tener las unidades y las alarmas, aparecen varias dimensiones auxiliares necesarias para dar soporte al mencionado patrón. A continuación se ilustran las tablas de dimensiones derivadas de la aplicación de este patrón y se hace una descripción de cada uno de sus atributos:

Dimensión *dim_unittype:*

Representa los tipos de unidades. Los tipos de unidades están indefinidos por lo cual se hace necesario su modelación en una entidad aparte.

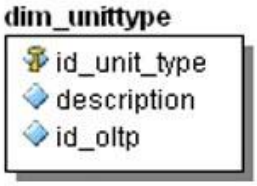


Figura 20: Dimensión dim_unittype.

- *id_unit_type:* Representa el identificador del tipo de unidad.
- *description:* Descripción textual del tipo de unidad.
- *id_oltp:* Representa el identificador que tenía el registro en el sistema OLTP.

Dimensión *dim_attributeunittype:*

Representa los atributos asociados a un tipo de unidad. Los atributos asociados a los tipos de unidades están indefinidos por lo cual se hace necesario su modelación en una entidad aparte.

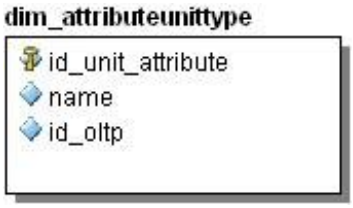


Figura 21: Dimensión dim_attributeunittype.

- *id_unit_attribute:* Representa el identificador del atributo.
- *name:* Nombre del atributo.
- *id_oltp:* Representa el identificador que tenía el registro en el sistema OLTP.

Dimensión *dim_valueunitattribute:*

Representa valores asociados a atributos de un tipo de unidad determinada.

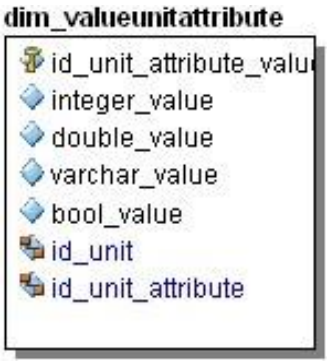


Figura 22: Dimensión dim_valueUnitAttribute.

- *id_unit_attribute_value:* Representa el identificador del valor.
- *integer_value:* Representa un valor entero de un atributo.
- *double_value:* Representa un valor real de un atributo.
- *varchar_value:* Representa un valor de texto de un atributo.
- *Bool:* Representa un valor booleano de un atributo.

Dimensión *dim_attributealarm*

Representa los atributos asociados a un tipo de alarma. Los atributos asociados a los tipos de alarmas están indefinidos por lo cual se hace necesario su modelación en una entidad aparte.

Figura 23: Dimensión dim_attributealarm.

- *id_alarm_attribute:* Representa el identificador del atributo.
- *name:* Nombre del atributo.
- *id_oltp:* Representa el identificador que tenía el registro en el sistema OLTP.

Dimensión *dim_valuealarmattribute:*

Representa valores asociados a atributos de un tipo de alarma determinada.

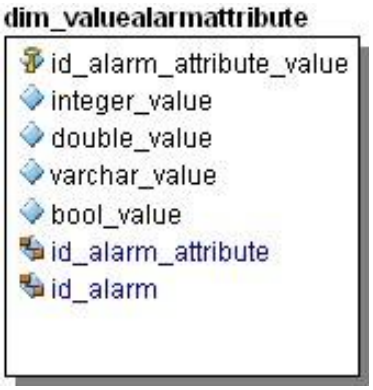


Figura 24: Dimensión dim_valueAlarmAttribute.

- *id_alarm_attribute_value:* Representa el identificador del valor.
- *integer_value:* Representa un valor entero de un atributo.
- *double_value:* Representa un valor real de un atributo.
- *varchar_value:* Representa un valor de texto de un atributo.
- *bool_value:* Representa un valor booleano de un atributo.

3.4.3 Tablas de hechos

Luego de definir las tablas de dimensiones corresponde definir las tablas de hechos. De acuerdo al modelo de constelación se definirán las tablas de hechos teniendo en cuenta los requisitos de información identificados.

En el modelo conceptual se identificaron dos relaciones:

Relación *Localización*

- La nueva tabla de hechos tendrá el nombre de *fact_localization.*
- Su clave principal será la combinación de las claves principales de las tablas de dimensiones *dim_group, dim_unitinfo, dim_geography, dim_time, dim_flote, dim_unit,* y un valor único representado por el atributo *subrogate_key.*
- Esta tabla de hechos tiene como principal función mantener los registros históricos de localización.

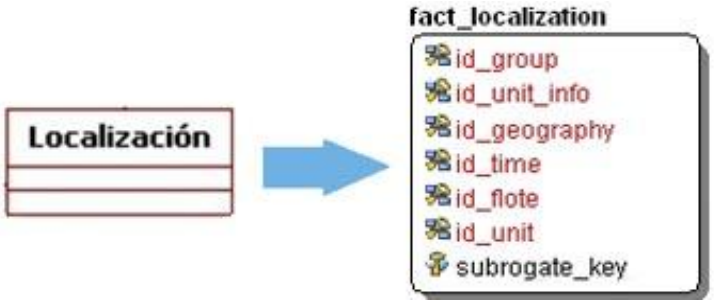


Figura 25: Mapeo de la relación Localización a la tabla de hechos fact_localization.

Relación *Alarma Activada*

- La nueva tabla de hechos tendrá el nombre de *fact_alarmtriggered.*
- Su clave principal será la combinación de las claves principales de las tablas de dimensiones *dim_flote, dim_group, dim_grate, dim_rute, dim_time, dim_geography, dim_unit, dim_alarm,* y un valor único representado por el atributo *subrogate_key.*
- Esta tabla de hechos tiene como principal función mantener los registros históricos de localización.

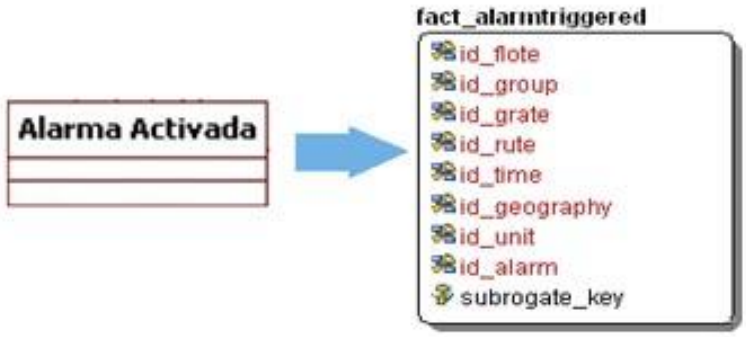


Figura 26: Mapeo de la relación Alarma Activada a la tabla de hechos fac_alarmTriggered.

3.4.4 Uniones

A continuación se presenta el diseño lógico del almacén de datos resultante de los procesos anteriores:

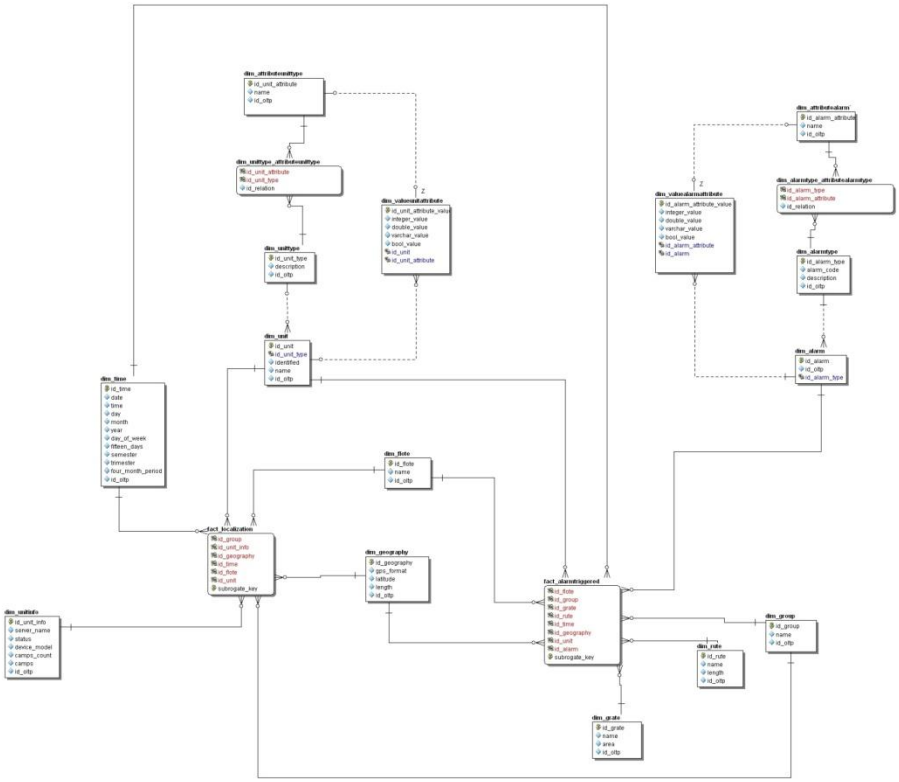


Figura 27: Modelo lógico del almacén de datos.

3.5 Integración de datos

Luego de haber construido el modelo lógico se procederá a poblar el almacén de datos con los datos que se encuentran en el sistema OLTP, para esto se utilizarán técnicas de extracción, transformación y carga de datos (procesos ETL). En este apartado se definirán los procesos ETL utilizados para realizar la carga inicial y la actualización de los datos mediante la herramienta Spoon, perteneciente a la suite de Pentaho Data Integration. Teniendo en cuenta que los procesos ETL encargados de la carga inicial y la actualización del almacén de datos son los mismos, estos se describirán en un mismo apartado.

3.5.1 Carga inicial y actualización del almacén de datos

A continuación se describe solamente la tarea diseñada para cargar inicialmente y actualizar de manera periódica el almacén de datos. Para obtener la descripción de todas las tareas y transformaciones diseñadas para cargar inicialmente y actualizar de manera periódica el almacén de datos, consulte el Anexo 4 de la versión digital.

Tarea: Poblar DWH

La tarea: Poblar DWH, describe todo el flujo necesario para poblar con los datos del sistema OLTP el almacén de datos. En caso de que el almacén esté vació, esta tarea llenará cada dimensión y cada tabla de hechos con todos los datos relevantes presentes en el sistema OLTP, realizando de esta forma la carga inicial del almacén. Si el almacén ya estuviera cargado con datos, esta tarea agregará los datos que hayan sido agregados al sistema OLTP y actualizará los valores de aquellas dimensiones que hayan sido modificadas desde la fecha y hora de término de la última carga al almacén de datos.

Figura 28: Tarea: Poblar DWH.

Conclusiones

En el presente capítulo se ha aplicado la metodología HEFESTO para realizar el diseño del almacén de datos que utilizará el módulo de reportes del sistema FE AVL para generar los diferentes tipos de reportes y elaborar los informes. Primeramente se determinaron los requisitos de información del cliente, luego se elaboró un modelo conceptual basado en estos requisitos que posteriormente derivaría en el diseño del almacén. Por último se diseñaron e implementaron los procesos de extracción, transformación y carga que poblarán inicialmente y mantendrán actualizado el almacén de datos.

CAPÍTULO 4: DISEÑO DEL SISTEMA

Introducción

En el presente capítulo se verán los aspectos correspondientes al diseño del módulo de reportes. Primeramente se hará la descripción del patrón arquitectónico utilizado en la solución. Luego se describen los diferentes patrones de diseño utilizados y sus aplicaciones y posteriormente, consecuentemente con las funcionalidades identificadas, se presenta el diagrama de paquetes del diseño y las tarjetas CRC – Class, Responsability, Colaboration / Clase, Responsabilidad y Colaboración – del sistema.

4.1 Patrón de arquitectura utilizado

Los patrones de arquitectura definen una familia de sistemas en términos de un patrón de organización estructural. Los estilos arquitectónicos definen tanto un vocabulario de tipos de componentes y conectores, como un conjunto de restricciones sobre cómo combinar esos componentes y conectores.

4.1.1 Patrón MVC (Model View Controller / 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. [25]

Vista: Presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario. [25]

Controlador: Responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y, probablemente, a la vista. [25]

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.

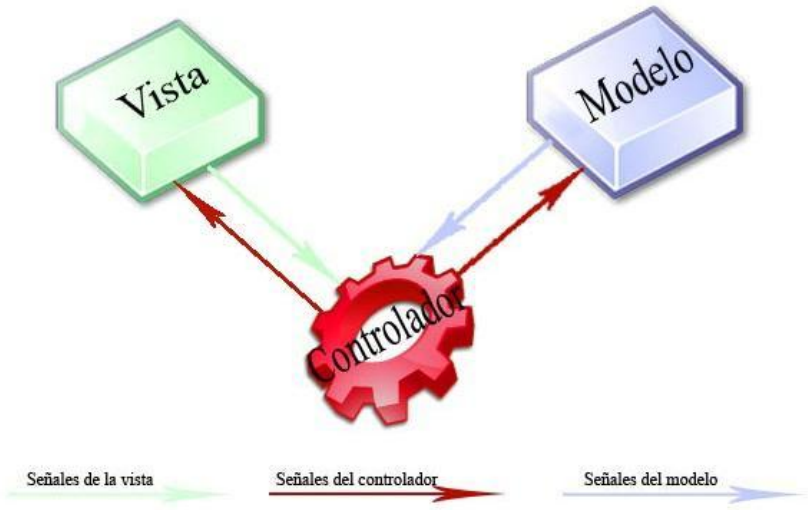


Figura 29: Diagrama del MVC del Sistema.

4.2 Patrones de diseño utilizados

Según Christopher Alexander[1], "cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema, de tal modo que se pueda aplicar esta solución un millón de veces, sin hacer lo mismo dos veces". Aunque Alexander se refería a patrones en ciudades y edificios, lo que dice también es válido para patrones de diseño orientado a objetos. [26]

Un patrón de diseño nomina, abstrae e identifica los aspectos caves de una estructura de diseño común, lo que los hace útiles para crear un diseño orientado a objetos reutilizable. [26]

4.2.1 Patrones GRASP

Los patrones GRASP describen los principios fundamentales del diseño de objetos y la asignación de responsabilidades, expresados como patrones. [27]

A continuación se exponen y ejemplifican los patrones GRASP utilizados en el diseño de la solución propuesta.

Experto en información

[1] Fue un eminente arquitecto. A él se atribuye la invención de los patrones de arquitectura en estructuras de construcción como ciudades y edificios.

Solución. Asignar una responsabilidad al experto en información -la clase que tiene la información necesaria para realizar la responsabilidad-. [27]

Problema. ¿Cuál es un principio general para asignar responsabilidades a los objetos?

Un Modelo de Diseño podría definir cientos o miles de clases software, y una aplicación podría requerir que se realicen cientos o miles de responsabilidades. Durante el diseño de objetos, cuando se definen las interacciones entre los objetos, tomamos decisiones sobre la asignación de responsabilidades a las clases software. Si se hace bien, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, y existen más oportunidades para reutilizar componentes en futuras aplicaciones. [27]

Ejemplo. En el siguiente caso la clase *D_Standar_Report* tiene la responsabilidad de conocer y establecer los atributos de clase *Title, Params* y *ReportTables.*

```
class D_Standar_Report : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString Title READ GetTitle WRITE SetTitle);
    Q_PROPERTY(QList<D_Param*> Params READ GetParams WRITE SetParams);
    Q_PROPERTY(QList<D_Report_Table*> ReportTables READ GetReportTables WRITE SetReportTables);
```

Figura 30: Ejemplo de aplicación del patrón Experto.

Creador

Solución. Asignar a la clase B la responsabilidad de crear una instancia de clase A si se cumple uno o más de los casos siguientes: [27]

- *B agrega* objetos de A.
- *B contiene* objetos de A.
- *B registra* instancias de objetos de A.
- *B utiliza más estrechamente* objetos de A.
- *B tiene los datos de inicialización* que se pasarán a un objeto de A cuando sea creado (por tanto, B es un Experto con respecto a la creación de A).

B es un *creador* de los objetos A. [27]

Si se puede aplicar más de una opción, inclínese por una clase B que *agregue o contenga* la clase A. [27]

Problema. ¿Quién debería ser el responsable de la creación de una nueva instancia de alguna clase? [27]

La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia, es útil contar con un principio general para la asignación de las

responsabilidades de creación. Si se asignan bien, el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización. [27]

Ejemplo. En el siguiente caso la clase *IReport_View* agrega objetos de tipo *QMenu,* luego en el constructor de dicha clase se instancian dichos objetos.

```
class IReport_View : public QMainWindow
{
    Q_OBJECT

private:
    QMenu * reportMenu;
    QMenu * fileMenu;
    QMenu * editMenu;
    QMenu * formatMenu;
    QMenu * designMenu;
    QMenu * viewMenu;
    QMenu * helpMenu;
```

```
IReport_View::IReport_View(QWidget *parent) :
    QMainWindow(parent)
{
    this->layoutMenu = new QHBoxLayout;

    //Creacion de menus
    this->fileMenu = new QMenu(tr("&Archivo"), this);
    this->editMenu = new QMenu(tr("&Edición"), this);
    this->reportMenu = new QMenu(tr("&Reporte"), this);
    this->viewMenu = new QMenu(tr("&Ver"), this);
    this->helpMenu = new QMenu(tr("&Ayuda"), this);
    this->formatMenu = new QMenu(tr("&Formato"), this);
    this->designMenu = new QMenu(tr("&Diseño"), this);
```

Figura 31: Ejemplo de aplicación del patrón Creador.

Bajo acoplamiento

Solución. Asignar una responsabilidad de manera que el acoplamiento permanezca bajo. [27]

Problema. ¿Cómo soportar bajas dependencias, bajo impacto del cambio e incremento de la reutilización? [27]

El *acoplamiento* es una medida de la fuerza con que un elemento está conectado a, tiene conocimiento de, confía en, otros elementos. Un elemento con bajo (o débil) acoplamiento no depende de demasiados otros elementos; "demasiados" depende del contexto, pero se estudiará. Estos elementos pueden ser clases, subsistemas, sistemas, etcétera. [27]

Una clase con alto (o fuerte) acoplamiento confía en muchas otras clases. Tales clases podrían no ser deseables; algunas adolecen de los siguientes problemas: [27]

- Los cambios en las clases relacionadas fuerzan cambios locales.
- Son difíciles de entender de manera aislada.
- Son difíciles de reutilizar puesto que su uso requiere la presencia adicional de las clases de las que depende.

Alta cohesión

Solución. Asignar una responsabilidad de manera que la cohesión permanezca alta. [27]

Problema. ¿Cómo mantener la complejidad manejable? [27]

En cuanto al diseño de objetos, la cohesión (o de manera más específica, la cohesión funcional) es una medida de la fuerza con la que se relacionan y del grado de focalización de las responsabilidades de un elemento. Un elemento con responsabilidades altamente relacionadas, y que no hace una gran cantidad de trabajo, tiene alta cohesión. Estos elementos pueden ser clases, subsistemas, etcétera. [27]

Una clase con baja cohesión hace muchas cosas no relacionadas, o hace demasiado trabajo.

Tales clases no son convenientes; adolecen de los siguientes problemas: [27]

- Difíciles de entender.
- Difíciles de reutilizar.
- Difíciles de mantener.
- Delicadas, constantemente afectadas por los cambios.

A menudo, las clases con baja cohesión representan un "grano grande" de abstracción, o se les han asignado responsabilidades que deberían haberse delegado en otros objetos. [27]

Controlador

Solución. Asignar la responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase que representa una de las siguientes opciones: [27]

- Representa el sistema global, dispositivo o subsistema (controlador de fachada).
- Representa un escenario de caso de uso en el que tiene lugar el evento del sistema (controlador de sesión o de caso de uso).
  - Utilice la misma clase controlador para todos los eventos del sistema en el mismo escenario de caso de uso.
  - Informalmente, una sesión es una instancia de una conversación con un actor. Las sesiones pueden tener cualquier duración, pero se organizan a menudo en función de los casos de uso (sesiones de casos de uso).

Problema. ¿Quién debe ser el responsable de gestionar un evento de entrada al sistema?

Un *evento del sistema* de entrada es un evento generado por un actor externo. Se asocian con *operaciones del sistema* -operaciones del sistema como respuesta a los eventos del sistema-, tal como se relacionan los mensajes y los métodos. [27]

Un *Controlador* es un objeto que no pertenece a la interfaz de usuario, responsable de recibir o manejar un evento del sistema. Un Controlador define el método para la operación del sistema. [27]

Ejemplo. En el siguiente caso se muestra un fragmento de código del método *ReceiveMessaje* de la clase *C_Report_Manager* la cual ejerce como controladora conectando, mediante este método, las señales provenientes de la vista con el modelo y viceversa.

```
void C_Report_Manager::ReceiveMessaje(QString pluginID,Element messaje)
{
    if(pluginID != PluginID())
       return ;

    QString accion(((QString*)(messaje["Accion"]))->data());
    if (accion == "NewDocument")
    {
        QString name(((QString*)(messaje["Name"]))->toStdString().c_str());
        QString content(((QString*)(messaje["Templete"]))->toStdString().c_str());
        C_Re                e::getInstance()->NewDocument(name, content);
    }
    else if (accion == "SetCurrentDocumentContent")
    {
        QString content(((QString*)(messaje["Content"]))->toStdString().c_str());
        C_Report_Core::getInstance()->SetCurrentDocumentContent(content);
    }
    else if (accion == "NewPlugin")
    {
```

Figura 32: Ejemplo de aplicación del patrón Controlador.

Fabricación Pura

Solución. Tal clase es una *fabricación* de la imaginación. Idealmente, las responsabilidades asignadas a esta fabricación soportan alta cohesión y bajo acoplamiento, de manera que el diseño de la fabricación es muy limpio, *o puro,* de ahí que sea una fabricación pura-. [28]

Finalmente, una fabricación pura implica construir algo, ¡lo que hacemos cuando estamos desesperados! [28]

Problema. ¿Qué objetos deberían tener la responsabilidad cuando no quiere violar los objetivos de Alta Cohesión y Bajo Acoplamiento, u otros, pero las soluciones que ofrece el Experto (por ejemplo) no son adecuadas? [28]

Los diseños orientados a objetos algunas veces se caracterizan por implementar como clases software las representaciones de los conceptos del dominio del mundo real para reducir el salto en la representación. Sin embargo, hay muchas situaciones en las que la asignación de las responsabilidades sólo a las clases software de la capa del dominio da lugar a problemas en cuanto a cohesión y acoplamiento pobres, o que el potencial para reutilizar sea bajo. [28]

Ejemplo. En el siguiente ejemplo la clase *D_BuildHTMLReport* constituye un ejemplo del patrón Fabricación Pura. Esta clase es un mero invento de la imaginación que representa un objeto abstracto, no identificado inicialmente dentro de los objetos del dominio, que cumple con las

responsabilidades de generar el código html necesario para construir de manera visual las tablas, líneas, secciones y demás elementos de los reportes.

```
class D_BuildHTMLReport : public QObject
{
    Q_OBJECT
    Q_PROPERTY(d_formatTable * TableFormat READ GetFormatTable WRITE SetFormatTable);

private:
    static D_BuildHTMLReport * instance; //Singleton
    explicit D_BuildHTMLReport(); //Constructor
    d_formatTable * tableFormat;

     //Disenno de tabla
    d_formatTable* GetFormatTable();
    void SetFormatTable(d_formatTable * tableFormat);

    //Metodos de insercion
    QString InsertSection(d_sectionFormat * section);
    QString InsertLine();
    QString InsertImage(QString filename);
    QString InsertTable(int columns, int rows);

     //Metodos publicos de disenno
    QString BeginBuild(QString sWebTitle = "", QString sCSS = "");
    QString BuildImage(QString HTML, QString sImagePath);
    QString BuildTitle(QString HTML, QString title);
    QString BuildParam(QString HTML, QString name, QString value);
    QString BegingBuildDataTable(QString HTML);
    QString BuildDataTableHeaders(QString HTML, QList<QString> headers);
    QString BeginBuildDataRow(QString HTML, int index);
    QString BuildSimpleDataRow(QString HTML, QString sValue, int index);
    QString FinishBuildDataRow(QString HTML);
    QString FinishBuildDataTable(QString HTML);
    QString FinishBuild(QString HTML);

 signals:

 public slots:

 };
```

Figura 33: Ejemplo de aplicación del patrón Fabricación Pura.

Indirección

Solución. Asigne la responsabilidad a un objeto intermedio que medie entre otros componentes o servicios de manera que no se acoplen directamente. El intermediario crea una *indirección* entre los otros componentes. [28]

Problema. ¿Dónde asignar una responsabilidad, para evitar el acoplamiento directo entre dos (o más) cosas? ¿Cómo desacoplar los objetos de manera que se soporte el bajo acoplamiento y el potencial para reutilizar permanezca más alto? [28]

4.2.2 Patrones GoF

El término GoF se refiere a “**G**ang **o**f **F**ouur” o “Pandilla de los cuatro”. Los patrones GoF son un grupo de veintitrés patrones descritos por cuatro autores en el popular libro “Patrones de Diseño”.

A continuación se describen y ejemplifican los patrones GoF utilizados en el diseño de la solución propuesta.

Singleton

Solución. Crear una clase que sea responsable de la creación de su única instancia interceptando las peticiones para crear nuevos objetos y que proporcione un método para el acceso a dicha instancia.

Problema. ¿Cómo garantizar que una clase solo tenga una instancia durante la ejecución del sistema? ¿Cómo disponer de un punto global de acceso a una clase?

Ejemplo. En el siguiente ejemplo la clase *C_Report_Core* es un *singleton,* o sea solamente tendrá una única instancia durante toda la ejecución del programa.

```
class C_Report_Core : public QObject
{
   private:
      static C_Report_Core * instance;
      explicit C_Report_Core(); //Constructor
   //Metodo que devuelve la instancia de la clase. Aplicación del patrón Singleton.
   static C_Report_Core* getInstance()
       {
          static QMutex mutex;

          if (!instance)
          {
              mutex.lock();
              if (!instance)
                  instance = new C_Report_Core();
              mutex.unlock();
          }

          return instance;
       }

 };
```

Figura 34: Ejemplo de aplicación del patrón Singleton.

Composite

Solución. Declarar una clase abstracta que represente tanto a primitivas como a objetos compuestos y que declare operaciones que serán compartidas por todos los objetos compuestos. Declarar clases que representen primitivas y clases que representen objetos compuestos. Estas clases heredaran de la mencionada clase abstracta.

Problema. ¿Cómo componer objetos en estructuras de árbol para representar jerarquías de parte-todo? ¿Cómo permitir que los clientes traten de manera uniforme a los objetos individuales y a los compuestos?

Ejemplo. En el siguiente ejemplo las cuatro clases que se ilustran forman un *Composite* u objeto compuesto. La clase *D_Report_Table* contiene una lista de *D_Table_Value,* clase ancestro de *D_Report_Table,* por cuanto se forma una relación recursiva que tiene como caso base la clase *D_Simple_Value*.

```
class D_Standar_Report : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString Title READ GetTitle WRITE SetTitle);
    Q_PROPERTY(QList<D_Param*> Params READ GetParams WRITE SetParams);
    Q_PROPERTY(QList<D_Report_Table*> ReportTables READ GetReportTables WRITE SetReportTables);
    Q_PROPERTY(bool isChartReport READ IsChartReport);

protected:
   QString title;
   QList<D_Param*> params;
   QList<D_Report_Table*> reportTables;
   bool bIsChartReport;
```

```
class D_Report_Table : public D_Table_Value
{
Q_OBJECT
Q_PROPERTY(QList<QList<D_Table_Value*> *> Table READ GetTable WRITE SetTable);
Q_PROPERTY(QList<QString> Headers READ GetHeaders WRITE SetHeaders);
private:
    QList<QString> headers;
    QList<QList<D_Table_Value*> *> table;
```

```
class D_Table_Value : public QObject
{
    Q_OBJECT
    Q_PROPERTY(bool isSimpleData READ IsSimpleData);

protected:
   bool bIsSimpleData;
```

```
class D_Simple_Value : public D_Table_Value
{
    Q_OBJECT
    Q_PROPERTY(QString Value READ GetValue WRITE SetValue);

private:
```

Figura 35: Ejemplo de aplicación del patrón Composite.

Command

Solución. Encapsular una petición en un objeto, permitiendo así parametrizar a los clientes con diferentes peticiones.

Problema. ¿Cómo enviar peticiones a los objetos sin saber nada acerca de la operación solicitada o de quien es el receptor de la petición?

Ejemplo. En el siguiente ejemplo se pone de manifiesto la aplicación del patrón command. La acción *newReport* es creada, luego *fileMenu* y *fileToolbar,* dos objetos de carácter diferente se subscriben a dicha acción. Por último, la acción es conectada a un controlador de eventos llamado *NewDocument()* el cual de desencadenará ya sea si se ejecuta la acción desde la barra de menú, como si se ejecuta desde la barra de herramientas.

```
this->newReport = new QAction(QIcon(":/iconos/document-new.png") ,tr("Nuevo reporte ..."), this->fileMenu);
this->fileMenu->addAction(this->newReport);
this->fileToolbar->addAction(this->newReport);
connect(this->newReport, SIGNAL(triggered()), this, SLOT(NewDocument()));
```

Figura 36: Ejemplo de aplicación del patrón Command.

4.3 Diagrama de paquetes del diseño

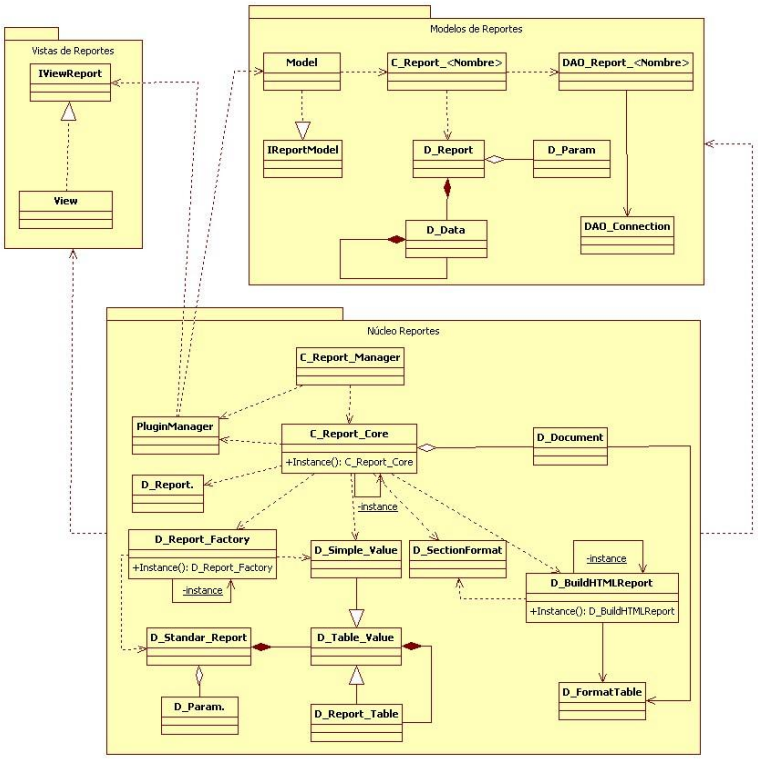


Figura 37: Diagrama de paquetes del Sistema.

4.4 Tarjetas CRC del sistema

A continuación se presentan y describen las clases del sistema mediante tarjetas *CRC – Class, responsibility and collaboration –*

Clase: C_Report_Manager	
Responsabilidad: ➢ Clase controladora principal. Maneja las peticiones de la vista principal del módulo de reportes, delegando las responsabilidades sobre las principales clases del modelo y devolviendo respuestas a la vista en el caso correspondiente.	**Colaboraciones:** ➢ C_Report_Core ➢ I_ReportModel ➢ I_ViewReport ➢ PluginManager

Clase: C_Report_Core	
Responsabilidad: ➢ Gestiona la lista de documentos activos. Además se encarga de estructurar los datos para su presentación.	**Colaboraciones:** ➢ D_Document ➢ D_Report ➢ D_Simple_Value ➢ D_Report_Factory ➢ D_BuildHTMLReport ➢ D_SectionFormat

Clase: D_BuildHTMLReport	
Responsabilidad: ➢ Clase de utilidad encargada de construir los diferentes objetos html que estructuran la presentación de los reportes.	**Colaboraciones:** ➢ D_FormatTable ➢ D_SectionFormat

Clase: D_Data	
Responsabilidad:	**Colaboraciones:**

➢ En una tabla de reporte, representa la unidad de información contenida en las celdas de dicha tabla. Este dato puede ser simple, como una cadena de texto o un valor entero, o puede ser otra tabla de datos.	-

Clase: D_Document	
Responsabilidad: Documento en el cual se pueden generar múltiples reportes. Encapsula los siguientes atributos: ➢ *DocName*: Nombre. ➢ *Content:* Contenido en formato html. ➢ *TableFormat:* Formato de las tablas del documento.	**Colaboraciones:** ➢ D_Standar_Report

Clase: D_FormatTable	
Responsabilidad: Formato de las tablas de datos. Encapsula los siguientes atributos: ➢ *HeaderTablesColour:* Color de los encabezados. ➢ *FirstBodyTablesColour:* Color de las filas impares. ➢ *SecondBodyTablesColour:* Color de las filas pares. ➢ *FontHeaderColor:* Color de la fuente de los encabezados. ➢ *FirstBodyFontColor:* Color de la fuente de las filas impares de los encabezados. ➢ *SecondBodyFontColour:* Color de la	**Colaboraciones:** -

fuente de las filas pares de los encabezados. ➢ *WidhtTablePercent:* Porciento del ancho con respecto al documento. ➢ *Padding:* Relleno de celda. ➢ *Margin:* Margen, en píxeles. ➢ *Secondary:* Determina se utilizará sólo el color de las filas impares para toda la tabla o si se utilizará también el color de las filas pares.	

Clase: D_Param	
Responsabilidad: Parámetro de un reporte. Encapsula los siguientes atributos: *Name:* Nombre del parámetro *Value:* Valor del parámetro.	**Colaboraciones:** -

Clase: D_Report	
Responsabilidad: Reporte proveniente de un plugin de reporte. Encapsula los siguientes atributos: ➢ *Title:* Título del reporte. ➢ *Params:* Lista de parámetros del reporte. ➢ *Headers:* Encabezados de la tabla de datos del reporte. ➢ *Data:* Datos del reporte.	**Colaboraciones:** ➢ IReport

Clase: D_Report_Factory	
Responsabilidad: Clase encargada de transforma un reporte proveniente de un plugin de reporte (*D_Report*)	**Colaboraciones:** ➢ D_Report ➢ D_Standart_Report

en un reporte manejable por el núcleo del sistema (*D_Standar_Report*).	➢ D_Simple_Value

Clase: D_Report_Table	
Responsabilidad: Tabla contenida en un reporte. Encapsula los siguientes atributos: ➢ *Headers:* Encabezado de la tabla. ➢ *Table:* Cuerpo de la tabla.	**Colaboraciones:** ➢ D_Table_Value

Clase: D_SectionFormat	
Responsabilidad: Formato de las secciones de los reportes. Encapsula los siguientes atributos: ➢ *Name:* Nombre de la sección. ➢ *Border:* Grosor en puntos del borde de la sección. ➢ *Paddign:* Relleno de la sección. ➢ *Widht:* Porciento de ancho de la sección con respecto al documento. ➢ *SectionColor:* Color de la sección. ➢ *SectionFontColor:* Color de la fuente de la sección.	**Colaboraciones:** -

Clase: D_Simple_Value	
Responsabilidad: Valor simple dentro de una tabla de reportes.	**Colaboraciones:** ➢ D_Table_Value

Clase: D_Standar_Report	
Responsabilidad: Reporte en el ámbito del núcleo del sistema. Encapsula los siguientes atributos:	**Colaboraciones:** ➢ D_Param ➢ D_Report_Table

➢ *Title:* Título del reporte. ➢ *Params:* Parámetros con los que fue configurado el reporte. ➢ *ReportTables:* Tablas con datos contenidas en dicho reporte.	

Clase: D_Table_Value	
Responsabilidad: ➢ Valor de una de datos. Este puede ser un D_Simple_Value o un D_Report_Table	**Colaboraciones:** -

Clase: DAO_Connection	
Responsabilidad: Establece la conexión con el almacén de datos.	**Colaboraciones:** **-**

Clase: C_<Nombre del plugin>	
Responsabilidad: Clase que se reimplementa para cada nuevo plugin de reporte y que se encarga de la construcción de los objetos D_Report y su devolución al sistema	**Colaboraciones:** ➢ D_Report ➢ DAO_<Nombre del plugin>

Clase: DAO_<Nombre del plugin>	
Responsabilidad: Clase que se reimplementa para cada nuevo plugin de reporte y que se encarga de recuperar del almacén de datos la información necesaria para la elaboración del reporte.	**Colaboraciones:** ➢ DAO_Connection

Clase: PluginManager

Responsabilidad:	**Colaboraciones:**
Gestiona la carga de los plugins de reportes de tipo vista y modelo para que puedan ser utilizados por el sistema.	➢ IViewReport ➢ IReportModel

Clase: IReportModel	
Responsabilidad:	**Colaboraciones:**
Interfaz de todos los plugins de reporte de tipo modelo.	➢ D_Data ➢ D_Report

Interfaz: IViewReport	
Responsabilidad:	**Colaboraciones:**
Interfaz de todos los plugins de reporte de tipo vista.	-

Interfaz: IViewReport	
Responsabilidad:	**Colaboraciones:**
Interfaz de todos los plugins de reporte de tipo vista.	-

Interfaz: IReport	
Responsabilidad:	**Colaboraciones:**
Interfaz de los objetos D_Report.	➢ D_Data ➢ D_Param

Interfaz: IReportModuleModel	
Responsabilidad:	**Colaboraciones:**
Interfaz del modelo del módulo de reportes.	-

Interfaz: IReportModuleView	
Responsabilidad:	**Colaboraciones:**

Interfaz de la vista del módulo de reportes.	-

Conclusiones

A lo largo del capítulo se vieron las cuestiones fundamentales correspondientes al diseño del módulo de reportes del sistema FE AVL. En primer lugar se vieron el patrón de arquitectura y los diferentes patrones de diseño empleados. Luego se presentó el diagrama de paquetes del diseño y por último se vieron las tarjetas CRC describiendo las principales clases, sus responsabilidades y sus colaboraciones con las demás clases del dominio.

CAPÍTULO 5: IMPLEMENTACIÓN Y PRUEBA

Introducción

En el siguiente capítulo se hará la descripción de los componentes del sistema y se mostrará cómo se relacionan estos a través del diagrama de componentes. Además se hará un análisis de cada subsistema en los cuales se realizarán una serie de pruebas, contribuyendo así a la calidad del sistema en general.

5.1. Diagrama de componentes del Sistema

Los diagramas de componentes describen los elementos físicos que forman un sistema y como se relacionan estos entre sí. A continuación se muestra el diagrama de componentes del módulo de reportes del Sistema FE AVL utilizando paquetes de clases a nivel lógico.

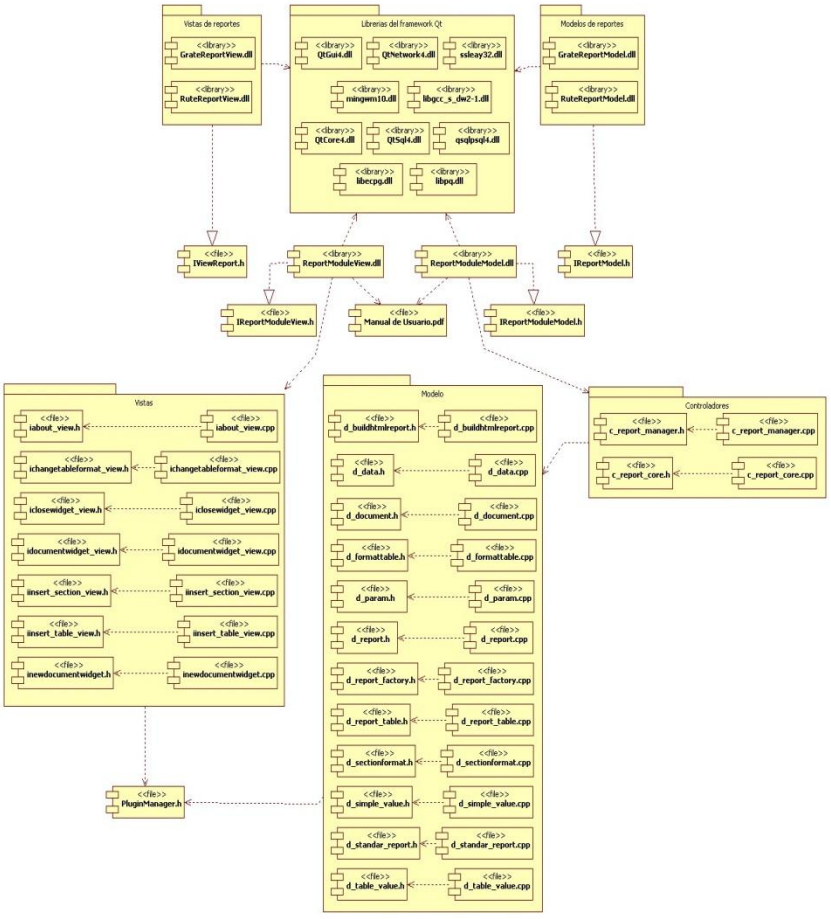


Figura 38: Diagrama de componentes del sistema.

5.1.1 Descripción de los paquetes y principales componentes

Paquete de componentes "Vista de reportes"

El paquete de componentes "Vista de reportes" se compone por librerías que conforman las vistas de los plugins de reportes.

Paquete de componentes "Modelo de reportes"

El paquete de componentes "Modelo de reportes" se compone por librerías que conforman el modelo de los plugins de reportes.

Paquete de componentes "Librerías del framework Qt"

El paquete de componentes "Librerías del framework Qt" se compone por las librerías del framework Qt que se utilizan en el sistema.

Paquete de componentes "Vistas"

El paquete de componentes "Vistas" está compuesto por las vistas o formularios utilizados en el sistema como interfaz de comunicación usuario-sistema.

Paquete de componentes "Modelo"

El paquete de componentes "Modelo" está compuesto por los ficheros de encabezado y los ficheros fuentes que representan el modelo del sistema.

Paquete de componentes "Controlador"

El paquete de componentes "Controlador" está compuesto por los ficheros de encabezado y los ficheros fuentes que se encargan de controlar el flujo de información entre las vistas y el modelo y viceversa.

Fichero PluginManeger.h

El fichero "PluginManager.h" es empleado para cargar en el módulo de reportes todos los reportes disponibles.

Ficheros IViewReport.h e IReportModel.h

Los ficheros “IViewReport.h” y “IReportModel.h” se emplean como lenguaje común para que se pueda establecer comunicación entre el los plugins de reportes y el módulo de reportes.

Ficheros IReportModuleView.h e IReportModuleModel.h

Los ficheros “IReportModuleView.h” y el fichero “IReportModuleModel.h” se emplean como lenguaje común para que se pueda establecer comunicación entre el módulo de reportes y el sistema FE AVL.

Librería ReportModuleView.dll

La librería “ReportModuleView.dll” encapsula todas las vistas y controles de usuario utilizados por el módulo de reportes como interfaz de comunicación usuario-sistema.

Librería ReportModuleModel.dll

La librería “ReportModuleModel.dll” encapsula todas las funcionalidades del sistema.

Manual de Usuario.pdf

El fichero “Manual de Usuario.pdf” es empleado para dar información acerca del manejo del sistema.

5.2 Diagrama de despliegue del Sistema

Un diagrama de despliegue representa como están distribuidos los diferentes nodos que componen un sistema. Los nodos son elementos físicos que cumplen una o varias funciones específicas dentro del entorno de un sistema computacional. A continuación se presenta el diagrama de despliegue del módulo de reportes:

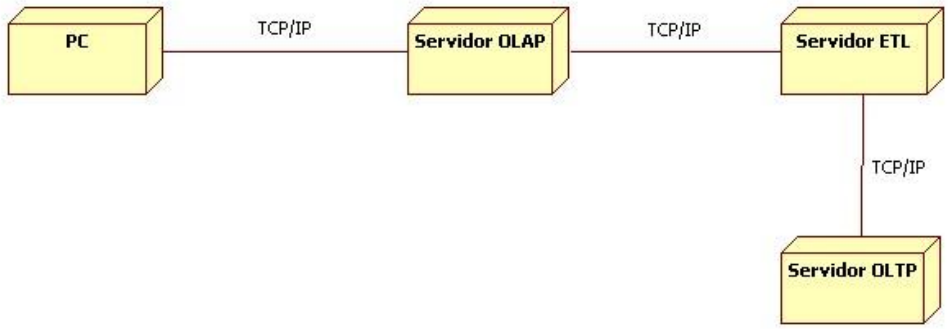


Figura 39: Diagrama de despliegue del sistema.

PC

Es una estación de trabajo de escritorio o portátil que tiene instalado el sistema FE AVL con el módulo de reportes integrado. Los requerimientos mínimos que debe poseer son:

- Memoria RAM con capacidad de 256 MB o superior.
- Microprocesador con capacidad de procesamiento de 1.6GHz o superior.
- Sistema operativo Windows Xp, Windows 7, Linux.

Servidor OLAP

Es un servidor que contiene el servidor de base de datos PostgresSQL 8.2 o superior donde está instalado el almacén de datos.

Servidor OLTP

Es un servidor de base de datos donde está instalada la base de datos operacional del sistema FE AVL.

Servidor ETL

Es un servidor donde está instalado el Pentaho Data Integration con la herramienta Spoon y donde están corriendo los procesos ETL.

5.3 Convenciones de archivos y paquetes

Se crean dos proyectos en Qt Creator: ReportModuleView y ReportModuleModel. Todas las clases y ficheros del sistema están agrupados de acuerdo a la estructura del lenguaje C++ y el framework Qt:

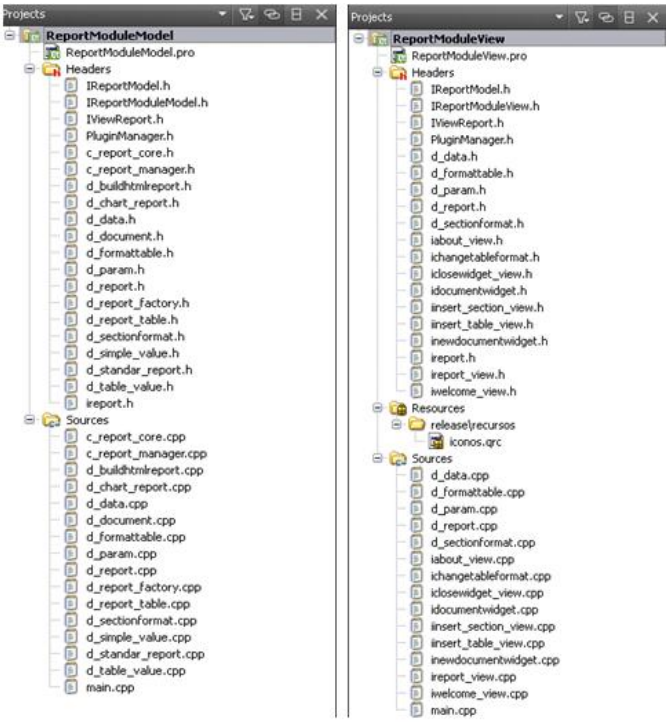


Figura 40: Convenciones de archivos y paquetes del módulo de reportes.

La estructura es la siguiente:

- **ReportModuleView.pro:** Archivo que contiene la información necesaria para la compilación de la librería ReportModuleView.dll.
- **ReportModuleModel.pro:** Archivo que contiene la información necesaria para la compilación de la librería ReportModuleModel.dll.
- **Headers:** Contiene las declaraciones de las clases, las cases estáticas y las interfaces.
- **Sources:** Contiene las implementaciones de las clases.
- **Resources:** Contiene los archivos de recursos utilizados.

5.4 Técnicas de programación

Durante la etapa previa al desarrollo del sistema, se realizó una investigación con el objetivo de identificar técnicas de programación que se pudieran emplear con el lenguaje de programación C++ y el framework Qt. Como resultado de esta investigación se decidió emplear la programación orientada a eventos, incorporada a través de macros por el framework Qt, como una extensión del lenguaje C++. Esta técnica permite que el sistema no tenga que estar constantemente buscando cambios en su estado sino que sean los cambios quienes le avisen al sistema. Otro aspecto que tiene un impacto positivo en el rendimiento del sistema es el uso frecuente de la estructura de datos

QMap ya que inserta y elimina elementos con una complejidad algorítmica de O log(n) haciendo un uso eficiente de la memoria. Además, teniendo en cuenta que la cantidad de plugins de reportes disponibles para su uso puede llegar a ser bastante grande, se diseño un mecanismo para cargar solo el plugin de reporte que se vaya a utilizar en un momento dado.

5.5 Pruebas

Las pruebas constituyen un aspecto esencial en el ciclo de desarrollo de software y son un elemento crítico para la garantía de la calidad del mismo. Estas representan la manera que tienen los desarrolladores y los clientes de constatar y comprobar el correcto funcionamiento de los sistemas.

5.5.1 Herramientas de pruebas

Durante todo el proceso de desarrollo se utilizaron algunas herramientas y dispositivos para realizarle pruebas al sistema.

Qt Creator

El *IDE* de desarrollo para Qt conjuntamente con C++ QtCreator constituyó una de las principales herramientas utilizadas para el *debuggeo* y la realización de pruebas al sistema. Para esto se empleo la clase del framework Qt QDebug la cual permite imprimir en una consola valores de punteros y variables, así como el análisis de condicionales y bucles.

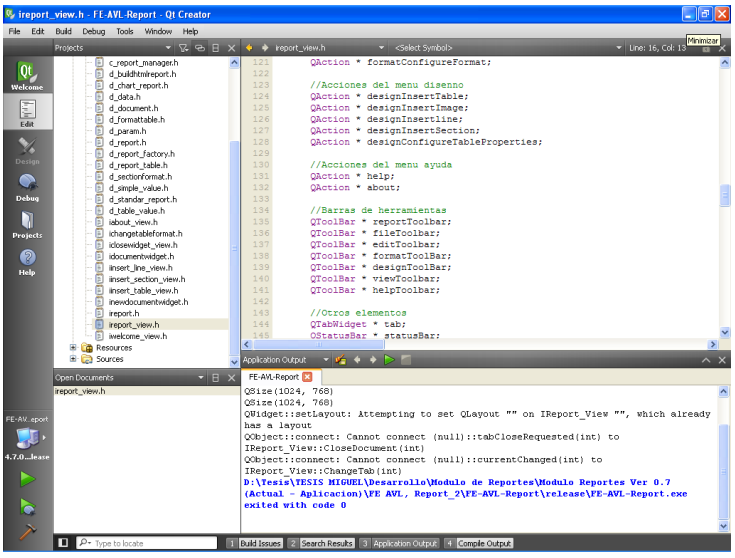


Figura 41: Qt Creator.

EMS

La herramienta EMS para PostgresSQL permitió la ejecución de consultas en el almacén de datos y el análisis de los resultados de dichas consultas, pudiéndose así comparar estos resultados con los resultados esperados en la ejecución de los reportes.

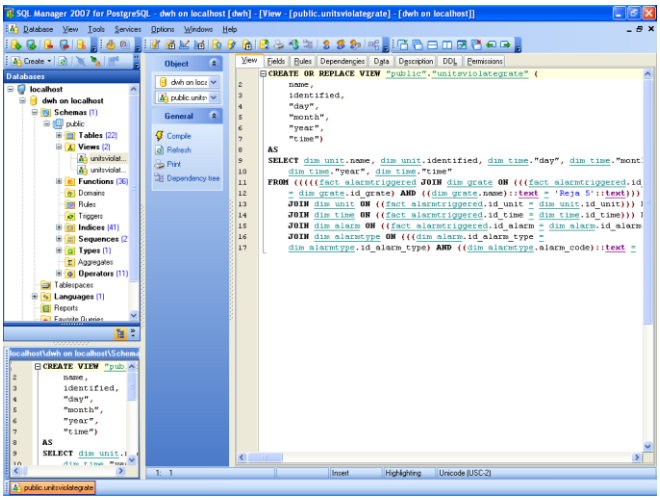


Figura 42: EMS para PostgresSQL.

Impresora EPSON LX-300

Mediante el uso de una impresora de marca EPSON, modelo LX-300 se pudieron probar las funcionalidades: Imprimir y Previsualizar Impresión.

5.5.2 Casos de pruebas

El método de caja negra para la realización de pruebas consiste en verificar que el sistema se comporta según los requerimientos funcionales definidos por el cliente. En este caso se necesitan para la realización de las pruebas la pila del producto y las historias de usuario definidas. Los casos de pruebas diseñados se encuentran descritos en el anexo 5 de la versión digital de este documento.

Conclusiones

En el presente capítulo se realizó el diagrama de componentes del sistema y la descripción de cada uno de los paquetes de componentes, así como de los principales componentes. Luego se analizó el diagrama de despliegue del sistema, y posteriormente se vieron las convenciones de archivos y paquetes y las principales técnicas de programación empleadas durante el desarrollo. Por último se vieron las herramientas utilizadas para la realización de pruebas al sistema y los casos de pruebas.

CONCLUSIONES GENERALES.

En el presente trabajo de diploma se desarrolló un sistema de reportes capaz de generar reportes históricos del sistema Front End AVL e integrarse a esta solución como parte de ella. De acuerdo a lo anteriormente dicho se concluye que tanto los objetivos específicos como el objetivo general de esta investigación se cumplieron satisfactoriamente. A continuación se incluyen una serie de recomendaciones a tener en cuenta para el desarrollo futuro de nuevas versiones del sistema.

RECOMENDACIONES.

- Se recomienda crear nuevos plugins de reportes a partir de los requisitos funcionales específicos de los clientes.
- Se recomienda, en nuevas versiones, mostrar reportes en gráficas de datos.
- Se recomienda, en nuevas versiones, la ampliación de las funcionalidades referentes al diseño de informes.

REFERENCIAS BIBLIOGRÁFICAS.

1. ¿Cómo funciona? [En línea] [Citado el 1 de diciembre de 2011.] (http://www.globalavl.eu/es/software-localizacion-gps/descripcion-servicio-localizacion/como-funciona-nuestro-software-localizacion.html)

2. Sistemas: FlotasNet [En línea] [Citado el: 1 de diciembre de 2011.] (http://www.fagorelectronica.com/flotas/flotasnet.html)

3. Sistemas: FlotasMap [En línea] [Citado el: 1 de diciembre de 2011.] (http://www.fagorelectronica.es/flotas/flotasmap.html)

4. Control y Gestión de Flotas [En línea] [Citado el: 1 de diciembre de 2011.] (http://www.vinasistemas.com/control-de-flotas.php)

5. Introducción a Qt [En línea] [Citado el: 1 de diciembre de 2011.] (http://www.zonaqt.com/tutoriales/tutorial-de-qt-4-por-zona-qt-0)

6. El lenguaje c++ [En línea] [Citado el: 1 de diciembre de 2011.] (http://www.articulandia.com/premium/article.php/12-03-2007El-Lenguaje-C.htm)

7. ¿Qué es Scrum? [En línea] [Citado el 6 de diciembre de 2011.] (http://www.proyectosagiles.org/que-es-scrum)

8. Rumbaugh, James; jacobson, Ivar y Booch, Grady. El Lenguaje Unificado de Modelado, Segunda Edición .2005 [Citado el: 2 de diciembre de 2011.]

9. Lenguaje de Modelado Unificado [En línea] [Citado el: 2 de diciembre de 2011.] (http://profejavaoramas.blogspot.com/2010/09/lenguaje-unificado-de-modelado-uml.html)

10. Start UML [En línea] [Citado el: 2 de diciembre de 2011.] (http://black-byte.com/review/staruml/)

11. IDE Qt Creator-Completo entorno de desarrollo multiplataforma [En línea] [Citado el: 2 de diciembre de 2011.] (http://pixelcoblog.com/qt-creator-completo-entorno-de-desarrollo-multiplataforma/)

12. Historia del PDF de Adobe [En línea] [Citado el: 2 de diciembre de 2011.] (http://www.adobe.com/es/products/acrobat/adobepdf.html)

13.Musciano, Chuck y Kennedy, Bill. HTML, La Guía Completa, Segunda Edición. 1999 [Citado el: 5 de diciembre de 2011.]

14. Lección 2: ¿Qué es HTML? [En línea] [Citado el: 5 de diciembre de 2011.] (http://es.html.net/tutorials/html/lesson2.php)

15.Graficos para representar datos [En línea] [Citado el: 5 de diciembre de 2011.] (http://www.desarrolloweb.com/articulos/875.php)

16. Almacén de Datos – Concepto [En línea] [Citado el: 6 de Diciembre de 2011.] (http://etl-tools.info/es/bi/almacendedatos_arquitectura.htm)

17. Esquema Estrella (Star Schema) [En línea] [Citado el: 6 de Diciembre de 2011.] (http://etl-tools.info/es/bi/almacendedatos_esquema-estrella.htm)

18. Esquema en Copos de Nieve [En línea] [Citado el: 6 de Diciembre de 2011.] (http://etl-tools.info/es/bi/almacendedatos_esquema-copo-de-nieve.htm)

19. Esquema de Constelación de Hechos (fact constellation schema) [En línea] [Citado el: 6 de Diciembre de 2011.] (http://etl-tools.info/es/bi/almacendedatos_esquema-constelacion.htm)

20. Redondo, P.M y Martín Moreno, J. Los Sistemas Inteligentes de Transporte, Pasado Reciente y Futuro [Citado el: 6 de Diciembre de 2011.]

21. MovilWeb: aplicación para el control de flota basada en la infraestructura de datos espaciales de la República de Cuba. [Citado el: 7 de Diciembre de 2011.] (http://biblioteca.universia.net/html_bura/ficha/params/title/movilweb-aplicacion-control-flota-basada-infraestructura-datos-espaciales-republica-cuba/id/44761251.html)

22. Ing. Bernabeu, Ricardo Darío. Hefesto V 2.0: Metodología para la construcción de un Data Werehouse. Córdoba, Argentina, 19 de Julio de 2010. [Citado el 5 de Marzo de 2012.]

23. PosgresSQL affiliates .ORG domain. [En línea] [Citado el: 5 de Marzo de 2012.] (http://www.computerworld.com.au/article/62894/postgresql_affiliates_org_domain)

24. Pentaho Data Integration (Integración de Datos Pentaho). [En línea] [Citado el: 5 de Marzo de 2012.]

25. Spain MVC [En línea] [Citado el: 11 de abril de 2012.] (http://www.spainmvc.com/)

26. Gamma, Erich; Helm, Richard; Johnson, Ralph y Vlissides, John. Patrones de Diseño. Elementos de software orientado a objetos reutilizable. Capítulo 1, Introducción. Addison Wesley. Edición española. 2002. Córdoba, España. [Citado el 11 de abril de 2012]

27. Larman, Craig. UML y Patrones. 2da Edición. Capítulo 16: GRASP: Diseño de objetos con responsabilidades. Prentice Hall. [Citado el 11 de abril de 2012]

28. Larman, Craig. UML y Patrones. 2da Edición. Capítulo 22: GRASP: Más patrones para asignar responsabilidades. Prentice Hall. [Citado el 11 de abril de 2012]

BIBLIOGRAFÍA.

Hernández Escalona, Yosel. Propuesta de Front End para gestión de posicionamiento vehicular sobre redes TETRA NEBULA. 2011

Kimbal, Ralph. El juego de Herramientas del Almacén de Datos. 1996.

Ing. Bernabeu, Ricardo Darío. Hefesto V 2.0: Metodología para la construcción de un Data Werehouse. Córdoba, Argentina, 19 de Julio de 2010.

Larman, Craig. UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado. 2da Edición. Prentice Hall.

Gamma, Erich; Helm, Richard; Johnson, Ralph y Vlissides, John. Patrones de Diseño. Elementos de software orientado a objetos reutilizable. Addison Wesley.

Conferencia #7: Introducción a los Almacenes de Datos, Modelo Multidimensional, Base de DatosII. http://eva.uci.cu

Fargor Electrónica, Sitio Oficial, Sistemas: Funcionalidades. http://www.fagorelectronica.es/flotas/funcionalidades.html

GLOSARIO DE TÉRMINOS.

AVL: Son la siglas de *Automatic Vehicle Location o Localización Automática de Vehículos.* Este concepto es aplicado a los sistemas de localización remota en tiempo real.

CRM: - *Customer Relationship Management / Administración basada en las relaciones con los clients* – Son sistemas informáticos de apoyo a la gestión de las relaciones con los clientes, a la venta y al marketing.

Data Mart: Tipo de almacén de datos que contiene la información de uno o unos pocos departamentos de una determinada institución o empresa.

Data Warehouse: Tipo de almacén de datos que contiene la información de todos los departamentos de una determinada institución o empresa.

ERP: - *Entreprise Resource Planning / Planificación de Recursos Empresariales* – Son sistemas de información gerenciales que integran y manejan muchos de los negocios asociados con las operaciones de producción y de los aspectos de distribución de una compañía en la producción de bienes o servicios.

Keyhole: Se refiere a la empresa pionera en el desarrollo de software especializado en aplicaciones de visualización de datos geoespaciales fundada en 2001 y adquirida por Google en 2004.

KML: -*Keyhole Markup Language / Lenguaje de Marcado Keyhole*-**:** Es un lenguaje de marcado desarrollado para representar datos geográficos en tres dimensiones.

Modelo de Datos: Es un lenguaje orientado a describir una Base de Datos. Un modelo de datos permite describir los elementos de la realidad que intervienen en un problema dado y la forma en que se relacionan esos elementos entre sí.

Modelo Dimensional: Es un modelo de datos en el cual una base de datos consiste en una sola tabla grande de hechos que son descritos usando dimensiones y medidas. Es una adaptación especializada del modelo relacional.

Modelo Relacional: Es un modelo de datos basado en la lógica de predicados y en la teoría de conjuntos. Es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Se base fundamentalmente en el uso de relaciones entre entidades.

OLAP: - *OnLine Analytical Process / Proceso Analítico en Línea* - es una solución utilizada en el campo de la inteligencia empresarial cuyo objetivo es agilizar el proceso de consultas en grandes bases de datos.

OLTP: - *OnLine Transaction Processing / Procesamiento de Transacciones en Línea* - es un tipo de sistema que facilita la administración de aplicaciones transaccionales, usualmente para entrada de datos y recuperación y procesamiento de transacciones.

Plugin: La palabra *plugin* se traduce en el vocabulario informático como complemento, conector o extensión. Un plugin es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica.

Release: Se refiere al acto de hacer público o poner a disposición de los usuarios finales un programa de software o cualquier otro tipo de información.

SAP: En alemán (*Systeme, Anwendungen und Produkte*). Es un conjunto de aplicaciones de negocios que provee integración de información, colaboración, funcionalidad específica de industria y escalabilidad.