Universidad de las Ciencias Informáticas

Facultad 5

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

Título: Módulo de Gestión y Archivo de Datos para el sistema SCADA.

Autor

Yusnier Manuel Sosa Vázquez.

Tutor

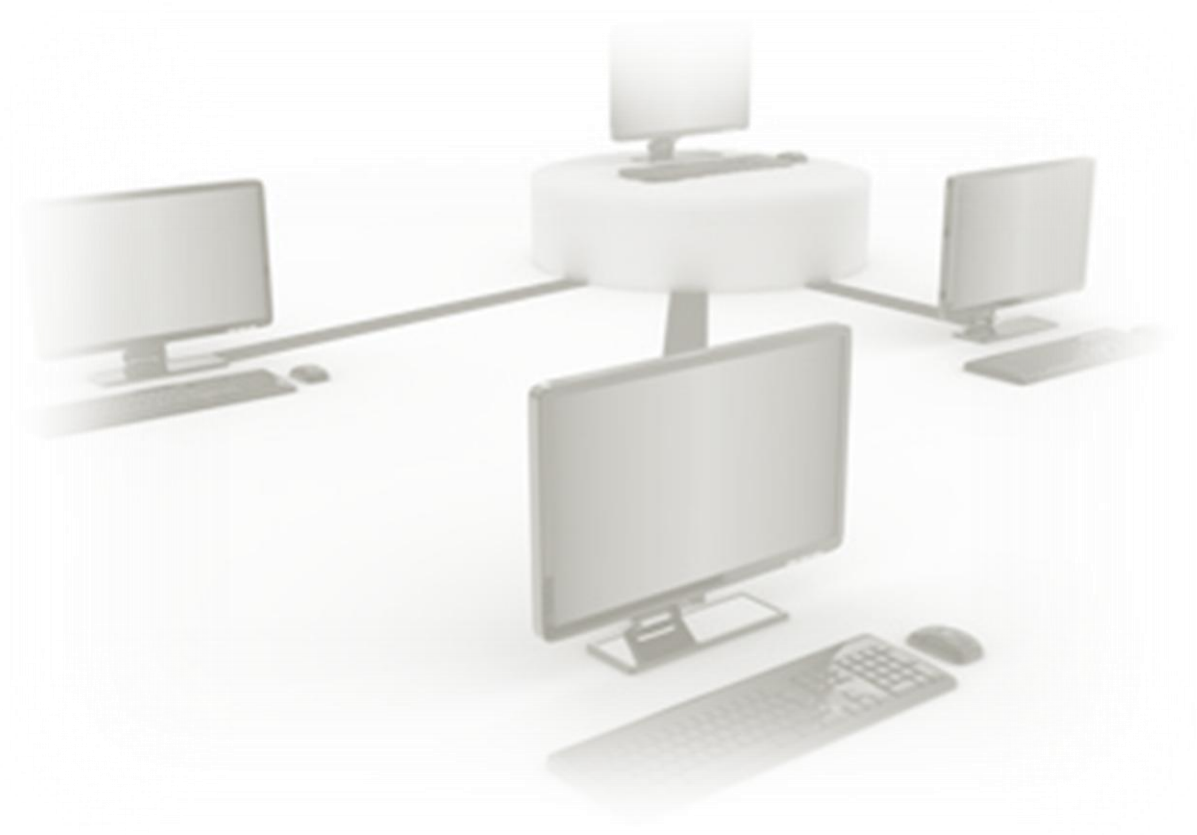
Ing. Abdelaziz De La Horra Díaz.

Co-tutor

Ing. José Antonio Aragón Cáceres

Ciudad de La Habana

Junio de 2010



"Si piensas que vales lo que sabes, estás muy equivocado. Tus conocimientos de hoy no tienen mucho valor más allá de un par de años. Lo que vales es lo que puedes llegar a aprender, la facilidad con la que te adaptas a los cambios que esta profesión nos regala tan frecuentemente."

José M. Aguilar

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Yusnier Manuel Sosa Vázquez	Abdelaziz De La Horra Díaz
_____	_____
Firma del Autor	Firma del Tutor

DATOS DE CONTACTO

Nombre y apellidos: Abdelaziz De La Horra Díaz.
Institución: Universidad de las Ciencias Informáticas (UCI).
Título: Ingeniero en Ciencias Informáticas.
e-mail: adelahorra@uci.cu

Líder de la línea de Persistencia del Centro de Informática Industrial, con tres años de experiencia en la materia de Base de Datos.

Nombre y apellidos: José Antonio Aragón Cáceres.
Institución: Universidad de las Ciencias Informáticas (UCI).
Título: Ingeniero en Ciencias Informáticas.
e-mail: jaaragon@uci.cu

Líder de la línea de Desarrollo de Comunicaciones del Departamento de Construcción de Componentes del Centro de Informática Industrial, con cuatro años de experiencia en el desarrollo de componentes de sistemas SCADA.

AGRADECIMIENTOS

A mi madre, por dedicarme su vida. A mi padre, por su ejemplo de dedicación y sacrificio desinteresado, en la vida no he conocido nadie igual. A mi hermana, por estar siempre a mi lado, y apoyarme en los buenos y malos momentos. A mi abuela por su amor y sus consejos, que me han ayudado a madurar. A mis amigos de la escuela, de los cuales he aprendido mucho. A todos mis profesores, por el conocimiento aportado a mi formación profesional. A todo aquel que ha contribuido a hacerme crecer como persona.

DEDICATORIA

A mis padres y a mi abuela.

RESUMEN

El presente trabajo de diploma se centra en el desarrollo de un módulo de Gestión y Archivo de Datos para el sistema SCADA, del Centro de Informática Industrial (CEDIN). El módulo permite la gestión de datos históricos, a través del desarrollo de extensiones o plugins, y el uso de un sistema de gestión de base de datos embebido. En el mismo se pueden encontrar aspectos generales acerca de los sistemas de control supervisor y de adquisición de datos (SCADA), de los sistemas de gestión de base de datos (SGBD) y sobre el concepto de complemento o extensión y sus aplicaciones en el área de la informática.

Como parte de la investigación se realizó un estudio sobre las principales funcionalidades que posee un módulo de Base de Datos de Históricos (BDH), como también se le conoce al módulo de Gestión y Archivo de Datos de un SCADA, y sobre los tipos de SGBD, con el objetivo de seleccionar el más factible. Se realizó la selección de un conjunto de tecnologías, adecuadas para el proceso de desarrollo. Se definieron los requerimientos funcionales y no funcionales que poseería el módulo, seguido por el modelado de la solución. Finalmente, se implementó el diseño definido y se realizaron pruebas a las funcionalidades críticas del sistema, arrojando resultados satisfactorios que validaron el uso de las tecnologías seleccionadas y el modelado propuesto.

Palabras claves:

BDH, SCADA, SGBD, embebido, plugin.

ÍNDICE

DECLARACIÓN DE AUTORÍA I
DATOS DE CONTACTO II
AGRADECIMIENTOS III
DEDICATORIA IV
RESUMEN V
ÍNDICE VI
ÍNDICE DE FIGURAS VIII
ÍNDICE DE TABLAS IX
INTRODUCCIÓN 1
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 5
1.1 Introducción 5
1.2 SCADA 5
1.2.1 Descripción 5
1.2.2 Funcionalidades 6
1.2.3 Estructura 7
1.3 Módulo de Base de Datos de Históricos. 8
1.3.1 Servicio histórico de variables. 8
1.3.2 Servicio histórico de alarmas. 11
1.3.3 Servicio histórico de eventos. 12
1.3.4 Servicio histórico de bitácoras. 12
1.3.5 Solicitud de información de los históricos de variables. 13
1.4 Base de datos. 14
1.4.1 Definición de Base de Datos. 15
1.4.2 Sistema de Gestión de Base de Datos (SGBD). 15
1.4.2.1 Objetivos. 15
1.4.2.2 Estructura general. 16
1.4.2.3 Ventajas y desventajas. 17
1.4.2.4 Servidor de Base de Datos Relacional. 19
1.4.2.5 SGBD Embebidos. 20
1.5 Complemento o Extensión. 23
1.6 Programación multihilos. 24
1.7 Conclusiones del capítulo. 25
CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA 26
2.1 Introducción 26
2.2 Propósito 26
2.3 Soluciones técnicas. 26
2.4 Tecnologías y herramientas a utilizar para el desarrollo. 26
2.4.1 UML como lenguaje de modelado. 27
2.4.2 RUP como metodología de desarrollo. 27
2.4.3 Visual Paradigm como herramienta CASE. 28
2.4.4 C++ como lenguaje de programación. 28
2.4.5 Eclipse como IDE de desarrollo. 29

2.4.6 Berkeley DB como SGBD Embebido. 29
2.4.7 Biblioteca Boost de C++ para el manejo de hilos y sistema de archivos. 31
2.4.8 Debian como sistema operativo. 31
2.5 Modelo de Dominio. 32
2.5.1 Clases Conceptuales o Glosario de términos del dominio. 32
2.6 Descripción del sistema propuesto. 33
2.6.1 Requisitos funcionales. 35
2.6.2 Requisitos no funcionales. 36
2.6.3 Actores del Sistema. 37
2.6.4 Casos de Uso del Sistema. 37
2.6.5 Diagrama de Casos de Uso del Sistema. 37
2.6.6 Descripción de los Casos de Uso del Sistema. 38
2.6.6.1 Registrar plugin. 38
2.6.6.2 Ejecutar servicio de plugin. 39
2.6.6.3 Detener servicio de plugin. 40
2.6.6.4 Eliminar plugin. 41
2.6.6.5 Listar plugins. 42
2.6.6.6 Consultar versión de plugin. 43
2.7 Conclusiones del capítulo. 44
CAPÍTULO 3: DISEÑO, IMPLEMENTACIÓN Y PRUEBAS 45
3.1 Introducción 45
3.2 Diseño del sistema. 45
3.2.1 Paquete App. 46
3.2.2 Paquete Configure. 47
3.2.3 Paquete InputCommand. 48
3.2.4 Paquete Message. 50
3.2.5 Paquete PluginsCore. 51
3.2.6 Paquete DBClient. 53
3.2.7 Subpaquete BerkeleyDB. 55
3.3 Implementación. 55
3.3.1 Estilo de codificación. 55
3.3.2 Diagrama de componentes. 59
3.4 Desarrollo de Plugins. 59
3.5 Pruebas. 64
3.6 Conclusiones del capítulo. 69
CONCLUSIONES 70
RECOMENDACIONES 71
REFERENCIAS BIBLIOGRÁFICAS 72
BIBLIOGRAFÍA CONSULTADA 73
ANEXOS 75
GLOSARIO DE TÉRMINOS Y ABREVIATURAS 89

ÍNDICE DE FIGURAS

Figura 1: Sistema SCADA. 6

Figura 2: Esquema de comunicación entre los diferentes módulos que intervienen en el proceso de recolección y archivo de históricos. 10

Figura 3: Esquema general de un SGBD. 15

Figura 4: Capas con las distintas API que provee Berkeley DB para el desarrollo. 30

Figura 5: Modelo de dominio. 32

Figura 6: Estructura general del módulo de Gestión y Archivo de Datos. 34

Figura 7: Diagrama de Caso de Usos del Sistema. 38

Figura 8: Paquetes del sistema. 46

Figura 9: Diagrama de Clases del paquete App. 47

Figura 10: Diagrama de Clases del paquete Configure. 48

Figura 11: Arquitectura en capas del paquete InputCommand. 49

Figura 12: Diagrama de Clases del paquete InputCommand. 49

Figura 13: Diagrama de Clases del paquete Message. 50

Figura 14: Diagrama de Clases del paquete PluginsCore. 52

Figura 15: Diagrama de Clases del paquete DBClient. 54

Figura 16: Diagrama de Clases del subpaquete BerkeleyDB. 55

Figura 17: Diagrama de componentes del sistema. 59

Figura 18: Estructura de paquetes para el desarrollo de plugins. 64

Figura 19: Esquema general del funcionamiento de plugin de pruebas plScadaPoints. 68

ÍNDICE DE TABLAS

Tabla 1: Descripción de los actores. 37

Tabla 2: Clasificación de los Casos de Uso del Sistema. 37

Tabla 3: Descripción del CUS “Registrar plugin”. 39

Tabla 4: Descripción del CUS “Ejecutar servicio de plugin”. 40

Tabla 5: Descripción del CUS “Detener servicio de plugin”. 41

Tabla 6: Descripción del CUS “Eliminar plugin”. 42

Tabla 7: Descripción del CUS “Listar plugins”. 43

Tabla 8: Descripción del CUS “Consultar versión de plugin”. 44

Tabla 9: Descripción de las clases del paquete App. 47

Tabla 10: Descripción de las clases del paquete Configure. 48

Tabla 11: Descripción de las clases del paquete InputCommand. 50

Tabla 12: Descripción de las clases del paquete Message. 51

Tabla 13: Descripción de las clases del paquete PluginsCore. 53

Tabla 14: Descripción de las clases del paquete DBClient. 54

Tabla 15: Descripción de los métodos abstractos de la interfaz IDataRecord. 61

Tabla 16: Descripción de los métodos abstractos de la interfaz IPlugin. 62

Tabla 17: Primera parte de las pruebas realizadas al sistema. 67

Tabla 18: Segunda parte de las pruebas realizadas al sistema. 68

INTRODUCCIÓN

En los inicios de la Automatización, los sistemas ofrecían capacidades muy simples para el monitoreo y control de procesos, y estaban desprovistos de cualquier función de aplicación. Durante el transcurso de las últimas décadas, la aparición de los ordenadores en el mundo de la industria ha contribuido de forma significativa al desarrollo de estos sistemas automáticos. Elementos de hardware cada día más potentes, la incorporación de nuevas funcionalidades, tanto en la adquisición como en la visualización de datos y el desarrollo de las redes industriales de comunicación, permiten actualmente realizar excelentes Sistemas de Automatización Industrial en tiempos mínimos.

La cumbre actual en la evolución de estos sistemas automáticos lo constituyen los sistemas de control supervisor y adquisición de datos (SCADA por sus siglas en inglés). Estos permiten supervisar y controlar las distintas variables que se encuentran en un proceso o planta determinada, utilizando para ello distintos periféricos, software de aplicación, unidades remotas, sistemas de comunicación, entre otros, posibilitando al operador, mediante la visualización en una pantalla de ordenador, tener un completo acceso al proceso.

En la Universidad de las Ciencias Informáticas (UCI), específicamente en la Facultad 5 radica el Centro de Informática Industrial (CEDIN), antecedido por el Polo de Hardware y Automática que contaba con su producto insignia: el sistema SCADA Nacional “Guardián del ALBA”, desarrollado en conjunto con personal venezolano a raíz del paro petrolero producido en la República Bolivariana de Venezuela a finales del año 2002. Este producto fue elaborado con la misión de controlar la producción de petróleo, utilizando para ello el software libre como directriz principal. El SCADA está conformado por diferentes módulos que se comunican entre sí, cada uno de ellos con una función específica.

Dentro de los subsistemas presentes en el SCADA se encuentra el módulo de Gestión y Archivo de Datos, más conocido como Base de Datos de Históricos (BDH), encargado del almacenamiento de los datos y cambios de estado dentro del sistema, posibilitando el análisis posterior de estos registros y la generación de reportes a distintos niveles. El módulo fue desarrollado acorde a los requisitos generales especificados para el SCADA en su concepción inicial, el cual estaba orientado específicamente al control de la producción de petróleo.

Con la experiencia adquirida en el Centro con el desarrollo del SCADA se incrementan las posibilidades de negocios, creando la necesidad de desplegarlo en varias entidades del país. Durante el proceso de integración del módulo de BDH, se han identificado nuevos escenarios que han provocado una alta variación con respecto a los requerimientos actuales. Dentro de los requerimientos presentes se encuentran:

- La incorporación y tratamiento de nuevos tipos de datos.
- El procesamiento en diversos modos de los datos a almacenar (esquemas de filtrado, métodos de compresión, entre otros).
- La eliminación de funcionalidades innecesarias en el funcionamiento del sistema a desplegar.
- El funcionamiento en diversas plataformas.

Debido al gran acoplamiento estructural que presenta el módulo de BDH con respecto al SCADA, este no posee la flexibilidad y extensibilidad necesarias para adaptarse a los cambios que imponen los diversos escenarios industriales cubanos, provocando que el mantenimiento del código fuente y el proceso de ajuste se hagan muy complejos, con un alto costo en tiempo y recursos.

Otro inconveniente, es que actualmente el módulo depende, para la gestión de información, de un Servidor de Base de Datos Relacional (SBDR), que aunque presenta ventajas en cuanto a capacidad física o posibilidades de comunicación cliente-servidor, cualquiera de los cambios estructurales mencionados pueden implicar modificaciones adicionales en el diseño de la base de datos usada por el módulo, incrementando aún más el costo en el proceso de ajuste. Además, el costo asociado de las comunicaciones entre procesos realizados por los SBDR, así como sus considerables requerimientos de memoria y procesamiento, hace que constituyan una solución poco eficiente en esquemas simples de almacenamiento, presentes fundamentalmente en empresas donde la disponibilidad de recursos de software y hardware es muy limitada y sólo requieren el manejo de muy pocos datos con bajos tiempos de acceso.

A partir de la **situación problemática** anteriormente expuesta surge como **Problema Científico** la siguiente interrogante: ¿Cómo proveer al SCADA de un módulo de Gestión y Archivo de Datos que se adapte a los requisitos presentes en los nuevos escenarios identificados?

Para darle respuesta a esta pregunta la presente investigación tiene como **Objeto de Estudio:** La Gestión de Datos en sistemas SCADA.

El **Objetivo General** de este trabajo es: Desarrollar un módulo de Gestión y Archivo de Datos que permita la modificación e introducción de nuevas funcionalidades de forma eficiente, y que pueda gestionar la información sin depender para ello de un SBDR.

Tomando como **Campo de Acción:** Los módulos de Gestión y Archivo de Datos en sistemas SCADA.

Para darle cumplimiento al objetivo planteado se proponen una serie de **Tareas de Investigación** relacionadas a continuación:

- Selección de las funcionalidades básicas que necesita un módulo de Gestión y Archivo de Datos.
- Selección de herramientas libres apropiadas y eficientes para el desarrollo.
- Identificación del tipo de SGBD más factible para su uso dentro del módulo.
- Selección del mecanismo más adecuado a los requerimientos, para la incorporación de extensiones funcionales en aplicaciones.
- Análisis y Diseño del módulo de Gestión y Archivo de Datos.
- Implementación del módulo de Gestión y Archivo de Datos.
- Desarrollo de una extensión funcional de pruebas que almacene variables históricas del SCADA.
- Realización de pruebas al módulo de Gestión y Archivo de Datos para validar su correcto desempeño.

En la realización de este trabajo se utilizan diferentes **métodos científicos** para estudiar las características del objeto de investigación:

Métodos Teóricos:

- Con el objetivo de extraer los rasgos que distinguen y caracterizan a los módulos de Gestión y Archivo de Datos en los sistemas SCADA, así como seleccionar los elementos más importantes relacionados con el objeto de estudio planteado se utilizó el método ***Analítico-sintético***.

- Con el objetivo de definir y representar gráficamente las ideas concebidas para el desarrollo del módulo de Gestión y Archivo de Datos se utilizó el método de ***Modelación***.

Métodos Empíricos:

- Con el objetivo de seleccionar la información necesaria en la investigación a partir del estudio de documentos y diferentes bibliografías se utilizó el método **Revisión de la documentación**.

Este trabajo está estructurado de la siguiente manera: resumen, introducción, tres capítulos de contenido, conclusiones, recomendaciones, referencias bibliográficas, bibliografía consultada, y anexos.

En el capítulo 1: Fundamentación Teórica, se describen las principales características de un SCADA, así como los servicios que brinda un módulo de BDH. Se hace un bosquejo de los conceptos fundamentales de BD, enfatizando en las particularidades de los SBDR y los SGBD Embebidos. Se dan nociones generales del concepto de Complemento o Extensión y de programación multihilos. En el capítulo 2: Características del Sistema, se ofrece una visión práctica del sistema, exponiéndose las herramientas y tecnologías usadas, así como los requisitos funcionales y no funcionales. Finalmente, se determinan los casos de uso y se describe cada uno. Por otra parte, en el capítulo 3: Diseño e Implementación, se diseña un sistema de clases, en correspondencia con las técnicas de la programación orientada a objetos, que luego son implementadas teniendo como resultado final un prototipo funcional del módulo. Por último, como parte de los anexos, se muestra un Glosario de Términos y Abreviaturas que facilita la comprensión del lenguaje técnico y las abreviaturas utilizadas en el trabajo.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se abordan conceptos esenciales para la comprensión del problema a resolver. Se describen de forma general los sistemas SCADA; se exponen las características de un Módulo de Base de Datos de Históricos, así como los distintos servicios que brinda. También se muestran aspectos relacionados con base de datos y sobre el concepto de complemento o extensión en aplicaciones.

1.2 SCADA

SCADA es el acrónimo de "*Supervisory Control And Data Acquisition*" (control supervisor y adquisición de datos). Los sistemas SCADA utilizan la computadora y tecnologías de comunicación para automatizar el monitoreo y control de procesos industriales. Estos sistemas son partes integrales de la mayoría de los ambientes industriales complejos o muy geográficamente dispersos, ya que pueden recolectar la información de una gran cantidad de fuentes muy rápidamente, y la presentan a un operador en una forma amigable. Los sistemas SCADA mejoran la eficacia del proceso de monitoreo y control proporcionando la información oportuna para poder tomar decisiones operacionales apropiadas. (1)

1.2.1 Descripción

Un sistema SCADA es una aplicación o conjunto de aplicaciones software especialmente diseñada para funcionar sobre ordenadores en el control de producción, proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, etc.) y controlando el proceso de forma automática desde la pantalla del ordenador. Además, provee de toda la información que se genera en el proceso productivo a diversos usuarios, tanto del mismo nivel como de otros supervisores dentro de la empresa: control de calidad, supervisión, mantenimiento, etc.

En este tipo de sistemas usualmente existe un ordenador, que efectúa tareas de supervisión y gestión de alarmas, así como tratamiento de datos y control de procesos. La comunicación entre los dispositivos de campo y la terminal central de procesamiento se realiza mediante buses especiales o a través de redes LAN. Todo esto se ejecuta normalmente en tiempo real, y están diseñados para dar al operador de planta la posibilidad de supervisar y controlar dichos procesos. (2)

Figura 1: *Sistema SCADA.*

1.2.2 Funcionalidades

El SCADA como sistema, comprende una serie funcionalidades encaminadas a establecer una comunicación lo más clara posible entre el proceso y el operador. Algunas de las principales prestaciones son las siguientes: (3)

- **Adquisición de datos:** Incluye recolectar, procesar, almacenar y mostrar la información recibida en forma continua desde los equipos de campo.

- **Supervisión:** El operador podrá observar desde el monitor la evolución de las variables de control, como cambios que se produzcan en la operación diaria de la planta.

- **Control:** El operador puede ejecutar acciones de control que podrán modificar la evolución del proceso en situaciones irregulares que se generen.

- **Generación de reportes:** De los datos adquiridos se pueden generar representaciones gráficas de los datos, predicciones, control estadístico, gestión de la producción, gestión administrativa y financiera.

1.2.3 Estructura

Sobre la base de las funcionalidades mencionadas se puede afirmar que un SCADA posee dos grandes grupos de componentes: uno de software y otro de hardware.

Entre los elementos de hardware pueden destacarse: (3)

- **Unidad terminal maestra (MTU):** Es el computador principal del sistema el cual supervisa y recoge la información del resto de las subestaciones; soporta una interfaz hombre máquina.

- **Unidad remota de telemetría (RTU):** Es un dispositivo instalado en una localidad remota del sistema y está encargado de recopilar datos para luego ser transmitidos hacia la Unidad Terminal Maestra.

- **Red de comunicación:** El sistema de comunicación es el encargado de la transferencia de información entre la planta y la arquitectura hardware que soporta el SCADA.

- **Instrumentación de campo:** Están constituidos por todos aquellos dispositivos que permiten tanto realizar la automatización o control del sistema.

Los componentes de software, también conocidos como módulos, varían un tanto en dependencia del sistema del que se trate. A pesar de ello, en casi todos pueden encontrarse los siguientes: (4)

- **Configuración:** Permite al usuario definir el entorno de trabajo de su SCADA, adaptándolo a la aplicación particular que se desea desarrollar.

- **Interfaz gráfico del operador:** Proporciona al operador las funciones de control y supervisión de la planta. El proceso se representa mediante sinópticos gráficos almacenados en el ordenador de proceso y generados desde el editor incorporado en el SCADA o importados desde otra aplicación durante la configuración del paquete.

- **Módulo de proceso:** Ejecuta las acciones de mando pre-programadas a partir de los valores actuales de variables leídas.

- **Gestión y archivo de datos:** Se encarga del almacenamiento y procesado ordenado de los datos, de forma que otra aplicación o dispositivo pueda tener acceso a ellos.

- **Comunicaciones:** Se encarga de la transferencia de información entre la planta y la arquitectura hardware que soporta el SCADA, y entre ésta y el resto de elementos informáticos de gestión.

1.3 Módulo de Base de Datos de Históricos.

En el enfoque distribuido de los sistemas SCADA, la recepción de información desde los niveles de campo hasta los niveles gerenciales, se perfila como el servicio más utilizado, resaltándose la captura y visualización, en tiempo real, en las consolas de operación. Sin embargo, en los sistemas donde se requiera un análisis de la información histórica capturada por los dispositivos de campo, así como de la sucesión de alarmas y eventos generados, se necesita un mecanismo que permita almacenar esos datos, partiendo de una configuración previa realizada por parte de los administradores del sistema.

La información almacenada es utilizada por una serie de aplicaciones entre las cuales se destacan los servicios gerenciales, como la gestión de producción, mantenimiento y control, utilizando algoritmos inteligentes, predictivos y adaptativos. La utilidad más inmediata es la generación de reportes por parte de los operadores y usuarios del SCADA. (5)

Existen varios tipos de históricos en la literatura de los sistemas SCADA, los más difundidos son los mencionados a continuación:

- Servicio de históricos de variables (puntos).

- Servicio de históricos de alarmas.

- Servicio de históricos de eventos.

- Servicio de históricos de bitácora.

1.3.1 Servicio histórico de variables.

Denominados por algunos autores como *Data Loggers*, estos servicios se especializan en el manejo de los históricos de las variables del sistema, sean estas variables de campo, memoria o calculadas.

Comúnmente este tipo de históricos se configuran para enviar al medio persistente el estado de una variable periódicamente o por excepción.

El término por excepción indica que la variable se envía a los históricos cuando se cumple alguna condición que permite decidir en qué instante debe almacenarse. Dentro de las excepciones más utilizadas por los sistemas SCADA se encuentran: (5)

- Ejecución de los históricos a una fecha y hora determinada.
- Envío a los históricos cuando el valor o calidad de la variable cambie.
- Envío a los históricos cuando se cumple una condición donde pueden intervenir varias variables y estados del sistema. Por ejemplo enviar al histórico la variable1 cuando la variable2 supere un valor determinado.

Los históricos de variables más utilizados son los que se configuran para ser almacenados periódicamente, el tiempo de envío al histórico de una variable depende de la dinámica de la misma y de los requerimientos específicos del proceso. También son utilizados mecanismos para enviar a los históricos sólo datos en forma de sumarios de la variable, lo que permite realizar una especie de compresión de los datos y optimizar los medios persistentes, por ejemplo el promedio en un período de tiempo configurable. Por otro lado, este servicio provee a los clientes la información de las variables en forma de series temporales, opcionalmente sumarizadas por intervalos de tiempo. (5)

La figura 2 muestra un esquema que relaciona los diferentes módulos que intervienen en el proceso de un historiador de procesos. En ella se marcan cinco niveles, especializados en funcionalidades que van desde la captura de la información hasta la persistencia de los datos.

1. **Nivel de Campo.** Está compuesto por dispositivos de campo (PLC, RTU, sensores, etc.) que contienen la información de proceso que requiere ser registrada para un análisis en tiempo real o histórico.
2. **Nivel de Recolección.** Se encarga de captar los datos del nivel de campo, respetando una lógica temporal o por eventos que es asociada a cada información requerida.

3. **Nivel de Base de Datos de Tiempo Real, BDTR.** Se encarga de recolectar los datos desde el nivel de recolección, procesarlos y servirlos a las aplicaciones de tiempo real, como por ejemplo las tendencias. Además, entrega los datos al nivel de manejo de históricos para garantizar la persistencia de los mismos. Este nivel comúnmente es redundante para garantizar la tolerancia ante fallos.

4. **Nivel de Historiador.** Se encuentra toda la lógica de recepción y envío de datos hacia los medios persistentes, ya sea por tareas periódicas o por excepción. Además, es el encargado de proveer la información histórica a aplicaciones como reportes, tendencias, etc. Este nivel también puede considerar la redundancia para garantizar un servicio estable libre de fallos.

5. El último nivel es el encargado de recepcionar la información del historiador, y almacenarlo de manera segura, utilizando opcionalmente servicios de replicación. En este nivel se acostumbra a utilizar servidores de bases de datos que permitan servicios de réplica y respaldo.

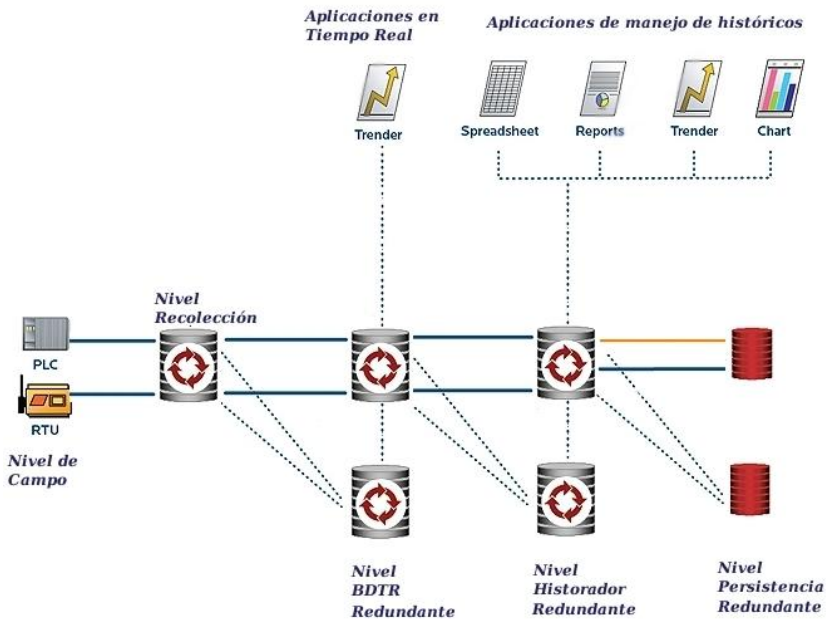


Figura 2: *Esquema de comunicación entre los diferentes módulos que intervienen en el proceso de recolección y archivo de históricos.*

En un SCADA, existen diversos mecanismos para organizar la información en los históricos, el más utilizado es el de organizar la información en un medio persistente que exprese los estados de las variables ordenadas por su estampa de tiempo, sin tener en cuenta criterios más complejos como los orientados a objetos.

En esquemas orientados a objetos, como los presentados en la especificación de históricos HDAIS (6), se organiza la información como una representación en modelo de objetos, los cuales contienen diferentes atributos, representados por las variables. Estos atributos contendrán las series temporales de las muestras. (5)

1.3.2 Servicio histórico de alarmas.

Este servicio se especializa en el manejo de los históricos de eventos anormales conocidos como ALARMAS. Las alarmas identifican eventos que indican mal funcionamiento del sistema, los cuales podrían conllevar a catástrofes. Las alarmas más comunes son asociadas a las variables del sistema, sean estas variables de campo o calculadas, dentro de estos tipos de alarmas se pueden mencionar, alarmas de nivel, tasa de cambio, entre otras.

Por otro lado, pueden existir alarmas asociadas a los dispositivos de campo del sistema (por ejemplo la falla de un PLC que contiene la información de variables del sistema) y alarmas de sistema (tales como la caída de una red o el fallo catastrófico de recursos de hardware). (5)

Los históricos de alarmas pueden ser divididos en dos partes fundamentales: (5)

- **Históricos de ocurrencia de alarmas**. La ocurrencia de alarmas tiene una importancia trascendental en el desempeño de un sistema, el correcto manejo de las mismas y mantener un registro detallado de las ocurrencias es un requisito de relevante importancia en cualquier sistema automatizado. Utilizando las correlaciones entre los valores de las variables (históricos de variables y estados actuales), y la información de los históricos de las ocurrencias de alarmas se puede llegar a predecir y peritar accidentes.

- **Históricos de seguimiento del manejo alarmas**. La ocurrencia de una alarma, indica un hecho que podría llevar a eventos catastróficos. El aviso de la ocurrencia de las alarmas a los

operadores, es la acción primaria que se lleva a cabo una vez que estas hayan sido detectadas. Sin embargo, el proceso sucesivo, en la mayoría de los casos, es responsabilidad del operador. Por ejemplo, una vez que el operador es avisado de la ocurrencia de una alarma, el primer paso que debe realizar es reconocer la existencia de la misma, posteriormente debe tomar acciones para restaurar el estado normal del sistema, como por ejemplo la modificación de algún parámetro de control.

1.3.3 Servicio histórico de eventos.

Se especializan en el manejo de los eventos que no son considerados catastróficos, pero que tienen impacto en el funcionamiento del sistema y que podrían tornarlo inestable. Dentro de estos se pueden mencionar autenticaciones, fallas en la ejecución de un comando, caídas temporales de la red, etc. Estos permiten las auditorías de los diferentes servicios del SCADA. (5)

Los eventos, se pueden definir como ocurrencias que pueden ser importantes para el análisis del comportamiento y seguridad del sistema. Llevar un registro de eventos, puede ser una herramienta que permite una corrección de los errores de implementación y huecos de seguridad del sistema.

Algunas de las categorías en las que se pueden clasificar los distintos tipos de eventos son:

- Eventos de Seguridad.
- Eventos de Operatividad.
- Eventos de Comunicación o RED.
- Eventos de Configuración.
- Eventos de Seguimiento de comandos.

1.3.4 Servicio histórico de bitácoras.

Estos servicios están orientados principalmente al almacenamiento de la información considerada relevante por los operadores, relativa a eventos que sucedan en su turno de trabajo. Permiten a los operarios documentar situaciones de operación que podrían ser de interés para ser consultadas por él o por los demás operadores en un futuro.

Esta información es utilizada por los operadores para buscar soluciones a situaciones que quizás otros operadores hallan documentado, y por los mantenedores para mejorar las aplicaciones, buscando mayor eficacia en el sistema. También es utilizado por los desarrolladores de software para proveer en versiones superiores soluciones a las necesidades de los usuarios.

Reutilizar las experiencias alcanzadas por los operadores, en cuanto al manejo del sistema, detección de errores, exposición de soluciones, entre otros, es uno de los servicios de históricos que se orienta a elevar la calidad operativa del sistema y que contribuye a realizar mejoras basadas en los criterios de los operadores. (5)

1.3.5 Solicitud de información de los históricos de variables.

Los servidores de históricos del estado de las variables de procesos, se caracterizan, principalmente, por proveer datos a los solicitantes para resolver tareas de manejo de tendencias de las series temporales de las variables. Sin embargo, también pueden ser solicitados por sistemas de reportes para hacer sumarios de los estados de variables de proceso, que comúnmente están relacionadas con los niveles gerenciales, como por ejemplo, producciones, mantenimientos, entre otros.

Las consultas solicitadas por los clientes al servicio de históricos de variables, se definen por un conjunto de parámetros que permiten seleccionar la información de los medios persistentes, dentro de ellos se pueden mencionar los siguientes: (7)

- ***StartTime:*** Especifica el comienzo del intervalo de tiempo en el que se desea hacer la consulta. Comúnmente si no se especifica este parámetro, se considera que es el instante de tiempo más antiguo disponible.

- ***EndTime:*** Especifica el fin del intervalo de tiempo en el que se desea hacer la consulta. comúnmente si no se especifica este parámetro, se considera que son todos los valores hasta el tiempo actual.

- ***Bounds:*** Por defecto las consultas toman todos los valores entre el *StartTime* y *EndTime*, si este parámetro está activo, indica que se tomarán todos los datos incluidos en el intervalo incluso aquellos que se corresponden con el *EndTime*, de otra forma no se incluirán los valores que se

corresponden con *EndTime*. Si este parámetro está activo y no existe un valor que se corresponda exactamente con el *StartTime*, se tomará el primer valor que se encuentre anterior al *StartTime*. De igual forma si no existe un valor exactamente en el *EndTime*, se ofrecerá el primer valor que se encuentre después del *EndTime*.

- ***Aggregate:*** Este parámetro está inspirado en la especificación OPC HDA (7), básicamente define los métodos para realizar sumarios de la información del punto. Lo agregados más utilizados son: promedio, mínimo y máximo en un rango determinado. Solamente se permite el cálculo de agregados sobre valores numéricos y se omiten los valores con calidad mala. En el tema de calidad, en dependencia de la aplicación del servidor, se pueden incluir los valores con calidad incierta. En el Anexo 1 de este documento puede encontrarse la tabla con la lista de los agregados especificados por OPC HDA.

- ***ResampleInterval:*** El servidor divide el intervalo de tiempo especificado por *StartTime* y *EndTime* en una secuencia de intervalos sobre los cuales se aplican los agregados. Este parámetro define la duración de esos intervalos.

- ***Query:*** Especifica la consulta que posibilita la selección de ciertos valores desde el intervalo especificado. El formato del query normalmente utilizado es SQL o XQuery.

1.4 Base de datos.

La importancia que tiene la información en la mayoría de las organizaciones y empresas ha llevado al desarrollo de gran cantidad de conceptos y técnicas para la gestión eficiente de los datos.

El término base de datos fue establecido por primera vez en 1963, en un Simposio celebrado en Santa Mónica, California, Estados Unidos. De forma sencilla, una base de datos no es más que un conjunto de información o de datos relacionados que se encuentran agrupados o estructurados. Con bastante frecuencia se trata con bases de datos manuales sin apenas notarlo, a una guía telefónica, los ficheros con los datos de los libros existentes en una biblioteca, un archivo que contiene recetas de cocteles, entre otras.

1.4.1 Definición de Base de Datos.

Desde el punto de vista informático, se define una base de datos como un conjunto exhaustivo de datos estructurados, fiables y homogéneos, organizados independientemente de su utilización e implementación en una computadora, accesibles en tiempo real, que pueden compartir varios usuarios con necesidades de información diferentes y no predecibles en el tiempo.

La idea general expresada en la definición anterior refiere el concepto de base de datos a una colección de datos estructurados independientemente de las aplicaciones y del soporte de almacenamiento que los contiene, que tenga la menor redundancia posible, y que además puedan ser compartidos con varios usuarios y/o aplicaciones, bajo un control centralizado. Generalmente, las bases de datos de las empresas requieren gran cantidad de espacio de almacenamiento en las computadoras; algunas alcanzan varios *gigabytes* y las más grandes hasta varios *terabytes*. (8)

1.4.2 Sistema de Gestión de Base de Datos (SGBD).

Los Sistemas de Gestión de Bases de Datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos y el usuario. Los Sistemas Gestores de Bases de Datos proporcionan un interfaz entre aplicaciones y sistema operativo, consiguiendo, entre otras cosas, que el acceso a los datos se realice de una forma más eficiente, más fácil de implementar y, sobre todo, más segura. (9)

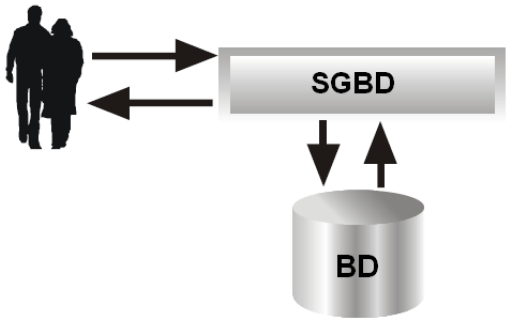


Figura 3: *Esquema general de un SGBD.*

1.4.2.1 Objetivos.

Dentro de los objetivos más importantes que debe cumplir un SGBD se encuentran: (10)

- Definir la Base de Datos mediante un Lenguaje de Definición de Datos.
- Separar la descripción y manipulación de los datos.
- Permitir la inserción, eliminación, actualización y consulta de los datos mediante el Lenguaje de Manejo de Datos.
- Gestionar la estructura física de los datos y su almacenamiento.
- Proporcionar un mecanismo de vistas, que permita a cada usuario tener su propia vista o visión de la base de datos.
- Eliminar la redundancia de datos.
- Proveer interfaces procedimentales y no procedimentales, permitiendo la manipulación por usuarios interactivos y programadores.
- Independizar la estructura de la organización lógica de los datos (Independencia física).
- Independizar la descripción lógica de la Base de datos y las descripciones particulares de los diferentes puntos de vistas de los usuarios.
- Permitir una fácil administración de los datos.

1.4.2.2 Estructura general.

Los elementos fundamentales que conforman un SGBD son:

- **Procesador-Administrador de Consultas**, traducción y chequeo de las consultas de los usuarios.
- **Administrador de transacciones**, debe facilitar un mantenimiento de las propiedades **ACID**:
    - ***Atomicity:*** Atomicidad. Asegura que una operación determinada se ha realizado o no, por tanto, ante un fallo, el sistema no puede quedar a medias.

- ***Consistency:*** Consistencia. El estado de la base de datos es consistente antes y después de cada transacción.
- ***Isolation:*** Aislamiento. Evita que una operación pueda afectar a otras, asegurando que en la realización de dos transacciones sobre la misma información, estas sean independientes y no generen ningún tipo de error.
- ***Durability:*** Durabilidad. Una vez realizada una operación los cambios comprometidos perduran en el tiempo.

- **Administrador de Almacenamiento*:*** Se encarga de administrar los archivos físicos de la base de datos y el buffer (memoria intermedia).

- **Repositorio** de metadatos y datos.

1.4.2.3 Ventajas y desventajas.

Los beneficios más importantes que aporta el uso de los SGBD son:

- **Mejora en la integridad de datos:** La integridad de la base de datos se refiere a la validez y la consistencia de los datos almacenados. Normalmente, la integridad se expresa mediante restricciones o reglas que no se pueden violar. Estas restricciones se pueden aplicar tanto a los datos, como a sus relaciones, y es el SGBD quien se debe encargar de mantenerlas.

- **Mejora en la seguridad:** La seguridad de la base de datos es la protección de la base de datos frente a usuarios no autorizados. Sin unas buenas medidas de seguridad, la integración de datos en los sistemas de bases de datos hace que éstos sean más vulnerables que en los sistemas de ficheros.

- **Mejora en la accesibilidad a los datos:** Muchos SGBD proporcionan lenguajes de consultas o generadores de informes que permiten al usuario hacer cualquier tipo de consulta sobre los datos, sin que sea necesario que un programador escriba una aplicación que realice tal tarea.

- **Mejora en la productividad:** El SGBD proporciona muchas de las funciones estándar que el programador necesita escribir en un sistema de ficheros. A nivel básico, el SGBD proporciona

todas las rutinas de manejo de ficheros, típicas de los programas de aplicación. El hecho de disponer de estas funciones permite al programador centrarse mejor en la función específica requerida por los usuarios, sin tener que preocuparse de los detalles de implementación de bajo nivel.

- **Mejora en el mantenimiento:** Los SGBD separan las descripciones de los datos de las aplicaciones. Esto es lo que se conoce como independencia de datos, gracias a la cual se simplifica el mantenimiento de las aplicaciones que acceden a la base de datos.

- **Aumento de la concurrencia:** La mayoría de los SGBD gestionan el acceso concurrente a la base de datos y garantizan que no ocurran problemas en el acceso de múltiples usuarios.

- **Mejora en los servicios de copias de seguridad y de recuperación ante fallos:** Los SGBD actuales funcionan de modo que se minimiza la cantidad de trabajo perdido cuando se produce un fallo.

Dentro de los inconvenientes a tener en cuenta al seleccionar un SGBD están:

- **Complejidad:** Los SGBD son por lo general software muy complejos, lo que obliga a los usuarios a tener un amplio dominio de todas sus funcionalidades para poder sacar un buen partido de ellos.

- **Tamaño:** Los SGBD son programas complejos y muy extensos que requieren una gran cantidad de espacio en disco y de memoria para trabajar de forma eficiente.

- **Costo del equipamiento adicional:** Los requisitos de hardware para correr un SGBD por lo general son relativamente altos, por lo que estos equipos pueden llegar a costar gran cantidad de dinero.

- **Prestaciones:** Los SGBD están escritos para ser más generales y útiles en muchas aplicaciones, lo que puede hacer que para algunas de ellas no sean tan rápidos. Si se manejan muy pocos datos que son usados por un único usuario por vez y no hay que realizar consultas complejas, entonces es posible que sea mejor usar algún sistema de archivos o una planilla de cálculo.

- **Vulnerable a los fallos:** El hecho de que todo esté centralizado en el SGBD hace que el sistema sea más vulnerable ante los fallos que puedan producirse.

- **Administración:** En la mayoría de los casos es necesario disponer de una o más personas que administren la base de datos, en la misma forma en que suele ser necesario en instalaciones de cierto porte, disponer de una o más personas que administren los sistemas operativos. Esto puede llegar a incrementar los costos de operación en una empresa.

1.4.2.4 Servidor de Base de Datos Relacional.

Un Servidor de Bases de Datos Relacional (SBDR) es un sistema bajo arquitectura cliente-servidor que proporciona servicios de gestión, administración y protección de la información a través de conexiones de red, gobernadas por unos protocolos definidos y a los que acceden los usuarios, de modo concurrente, a través de aplicaciones clientes, bien sean herramientas del propio sistema como aplicaciones de terceros.

Los servidores de base de datos solucionan los problemas de las empresas al manejar grandes volúmenes de información de una manera estable, fiable, coherente y segura en un entorno heterogéneo de trabajo y de necesidades de información. (11)

Algunos de los SGBD más importantes que ofrecen esta característica son:

Oracle: Surgido en la década del 70, gracias a un excepcional estudio sobre SGBD realizado por George Koch. Actualmente es uno de los sistemas de BD más demandados a nivel mundial, manteniendo una posición líder en el mercado de las BD relacionales. La gran potencia que ofrece y su elevado precio, hacen que se vea principalmente en empresas muy grandes y multinacionales, por norma general.

SQL Server: Es un SGBD relacional perteneciente a Microsoft, que se usa desde, en portátiles y ordenadores de sobremesa hasta en servidores corporativos. Se desarrolló originalmente en los años 80 en SyBase para sistemas UNIX y posteriormente pasado a sistemas Windows NT de Microsoft. Desde 1994 Microsoft ha lanzado versiones de SQL Server desarrolladas independientemente de SyBase, que dejó de utilizar el nombre SQL Server a finales de los años 90.

MySQL: SGBD muy conocido y ampliamente usado por su simplicidad y notable rendimiento. Aunque carece de algunas características avanzadas disponibles en otros SGBD del mercado, es una opción atractiva tanto para aplicaciones comerciales, como de entretenimiento, precisamente por su facilidad de uso y tiempo reducido de puesta en marcha. Esto y su libre distribución en Internet bajo licencia GPL le otorgan como beneficios adicionales contar con un alto grado de estabilidad y un rápido desarrollo.

PostgreSQL: Es un SGBD orientado a objetos (SGBDOO o ORDBMS en sus siglas en inglés) muy conocido y usado en entornos de software libre porque cumple los estándares SQL92 y SQL99, y también por el conjunto de funcionalidades avanzadas que soporta, lo cual lo sitúa al mismo a un mejor nivel que muchos SGBD comerciales.

1.4.2.5 SGBD Embebidos.

Un SGBD embebido es un sistema que está estrechamente integrado con una aplicación de software que requiere acceso a los datos almacenados, de manera que el sistema de base de datos queda oculto a los fines del usuario final que usa la aplicación. Normalmente, el proceso de integración se logra mediante la compilación o enlace de alguna biblioteca junto con la aplicación de software en la cual reside, realizando la gestión de los datos a través de interfaces de programación, que pueden incluir desde SQL como lenguaje de consulta, hasta el manejo de los tipos de datos nativos del lenguaje de programación utilizado por la interfaz.

En adición a las ventajas generales de los SGBD expuestos anteriormente también se pueden mencionar algunos beneficios asociados al uso de los SGBD embebidos:

- A diferencia de la arquitectura de comunicación cliente-servidor presente en los SGBD Relacionales, en la mayor parte de los SGBD embebidos, ambos, cliente y servidor corren juntos en el mismo proceso, reduciendo la latencia en el acceso a la base de datos, debido a que las llamadas a funciones dentro de un único proceso son más eficientes que la comunicación entre procesos independientes. Además, se hace innecesaria o no obligatoria la necesidad de configurar una red o su administración.

- Generalmente los SGBD embebidos proveen a las aplicaciones de software que los utilizan, un API (*Application Programming Interface*) con todas las funcionalidades necesarias para la

configuración, administración y gestión de datos. Esta característica reduce o evita completamente la necesidad de contar con personal adicional que administre o configure la base de datos, abstrayendo al usuario de todos estos procesos y limitándolo solamente a la interacción con la aplicación.

- El hecho de que este tipo de SGBD, se encuentre incrustado en la aplicación externa que lo utiliza, posibilita una mayor facilidad a la hora de desplegar el sistema, ya que no es necesario instalar el SGBD por separado.

- Los SGBD embebidos son herramientas muy configurables en aspectos relacionados con el uso de memoria, espacio en disco, nivel de concurrencia en el acceso a datos, severidad en la recuperación ante fallos, entre otras. Esto ofrece la ventaja de adecuar sus características a la de los requerimientos del sistema que lo alberga, de la forma más eficiente posible.

Estos sistemas son ideales en escenarios en los que se requiere el acceso a la base de datos por uno o muy pocos procesos, en aplicaciones en las que no se manejan grandes volúmenes de datos y se necesita un rápido acceso a los mismos, mediante consultas que no posean una elevada complejidad o algún método específico para la gestión de datos. Otra característica a valorar a la hora de seleccionar un SGBD embebido, es su nivel de robustez; la razón de este requisito radica en que habitualmente en su entorno de funcionamiento, no existe un administrador de base de datos, quedando de parte del sistema toda la responsabilidad de respuesta ante cualquier fallo que pueda ocurrir.

Dentro de los SGBD más conocidos de este tipo, se pueden mencionar los siguientes:

SQLite: Es un proyecto de dominio público creado por D. Richard Hipp, el cual implementa una pequeña librería de aproximadamente 500KB, programado en el lenguaje C, es totalmente libre y tiene como función hacer de un SGBD relacional. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un solo fichero estándar en la máquina host. La biblioteca implementa la mayor parte del estándar SQL-92, incluyendo transacciones y el manejo de *triggers*. En su versión 3, SQLite permite base de datos de hasta 2 terabytes de tamaño, y también permite la inclusión de campos tipo BLOB. Grandes empresas como Adobe, Firefox, Google, McAfee, Toshiba, Sun Microsystem,

Symbian y Microsoft hacen uso de SQLite para el desarrollo de mucho de sus productos, demostrando de esta manera la confianza y el gran rendimiento de la misma.

BerkelyDB: Fue desarrollado en la Universidad de Berkeley, California por la compañía Sleepycat Software, la cual fue adquirida en febrero de 2006 por Oracle Corporation. BerkeleyDB es una familia de bases de datos incorporadas, de código abierto, que permite a los desarrolladores incorporar en sus aplicaciones un motor de base de datos transaccional, rápido y escalable con disponibilidad y confiabilidad de clase industrial. BerkeleyDB es actualmente el motor de base de datos embebido más conocido del mundo, con probada fiabilidad y más de una década de uso en producción. Proporciona a los desarrolladores una gestión de datos con cero administraciones, a través de distintas interfaces: SQL con la introducción de manejadores para los estándares ODBC y JDBC, registros de tipo clave/valor, XML para la gestión de documentos de este tipo mediante el lenguaje XQuery, o la interfaz Java. Está incluido en la mayoría de los sistemas operativos actuales e incluso en sistemas operativos de tiempo real. También se encuentra disponible en diversos lenguajes de programación como C, C++, Java, C#, Perl, Python, Ruby, Tcl y muchos otros, con abundante documentación para el desarrollo.

IBM SolidDB: Originalmente propiedad de la compañía Solid Information Technology, fue adquirida en enero de 2008 por IBM. SolidDB es un SGBD híbrido, totalmente transaccional, que combina la gestión de datos tanto en disco como en memoria y es históricamente utilizado como SGBD embebido en equipos de telecomunicaciones, software de red, y sistemas similares.

eXtremeDB: Fue lanzado por la compañía McObject como la primera base de datos en memoria de código diseñado desde cero para sistemas embebidos de tiempo real. Cumple con las propiedades ACID, y su familia de productos incluye versiones para 64 bits con registro de transacciones y un prototipo de fusión hibrida que combina tanto el almacenamiento en disco como en memoria. En el 2008 McObject introduce la primera versión de eXtremeDB en modo de núcleo, constituyendo el primer SGBD diseñado para operar incrustado en el núcleo de un sistema operativo. Actualmente eXtremeDB es usado en millones de sistemas embebidos de tiempo real en todo el mundo.

1.5 Complemento o Extensión.

En el área de la Informática un complemento es una aplicación que se relaciona con otra para aportarle nuevas funcionalidades generalmente muy específicas. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de una interfaz de programación de aplicaciones (API). También se lo conoce como *plug-in* (del inglés "enchufable"), *add-on* (agregado), complemento, conector o extensión.

Los complementos permiten:

- A los desarrolladores externos colaborar con la aplicación principal extendiendo sus funciones.
- Personalizar la aplicación principal de maneras no pensadas por el autor.
- Reducir el tamaño de la aplicación.
- Separar el código fuente de la aplicación a causa de la incompatibilidad de las licencias de software.

De forma general el funcionamiento de los complementos se basa en una aplicación principal o host que proporciona servicios que el complemento puede utilizar, incluyendo un método para que se registren a sí mismos y un protocolo para el intercambio de información. Los complementos dependen de los servicios prestados por la aplicación que los acoge y no suelen funcionar por sí solos. Por el contrario, la aplicación principal funciona independientemente de ellos, lo que permite a los usuarios finales añadir y actualizar los complementos de forma dinámica sin necesidad de hacer cambios a la aplicación principal. Las APIs proporcionan una interfaz estándar que permite a terceros crear complementos que interactúan con la aplicación principal. Un API estable permite que complementos de terceros funcionen como la versión original y amplíen el ciclo de vida de las aplicaciones obsoletas. En la actualidad los desarrolladores suelen construir los plugins a través de bibliotecas dinámicas, las cuales son cargadas en tiempo de ejecución de la aplicación principal. Estas bibliotecas funcionan como contenedores, encapsulando objetos que implementan las distintas interfaces que brinda la aplicación para extender sus funcionalidades.

Algunos de los tipos de aplicaciones que suelen incluir plugins como mecanismo de extensión son:

- **Navegadores web**. Es frecuente requerir ciertos complementos que amplíen las funciones de las páginas web para ver contenidos interactivos, videos y otros elementos similares

- **Reproductores de música y video**. Algunos permiten añadir complementos para reproducir formatos que no son soportados originalmente, producir efectos de sonido o video, mostrar animaciones o visualizaciones que se mueven de acuerdo con la música que se está escuchando, entre otras opciones.

- **Sistemas de gestión de contenidos**. Permiten cambiar la apariencia, añadir botones u otro tipo de contenido a las páginas web que generan.

- **Juegos por computadora**. Las arquitecturas de numerosos juegos suelen utilizar complementos que permiten a los editores, ya sean los creadores originales o terceros, agregar características de diseño o funcionalidades tanto a los caracteres del propio juego como al entorno donde se desempeñan.

El uso de los complementos o plugins constituye una de las técnicas de mayor aceptación en la actualidad para la organización de grandes proyectos. Las dependencias son muy reducidas y es fácil trabajar en la sustitución de los sistemas específicos, en lugar del estancamiento de todo su proyecto o equipo hasta que la base del código haya sido completamente rediseñada.

1.6 Programación multihilos.

Un hilo (*thread*) es un flujo de control secuencial que ejecuta un segmento de código dentro de un programa (proceso). Un hilo normalmente comparte su memoria con otros hilos, a diferencia de un proceso real que generalmente posee su propio espacio de memoria. El hecho de que varios hilos compartan el mismo espacio de memoria dentro un proceso les permite acceder a recursos comunes como variables globales, archivos abiertos, señales y semáforos. A pesar de ello, cada hilo posee información propia que no comparte con otros, como son el contador de programa, la pila de ejecución (*stack*) y los registros.

En un entorno de multitarea basado en hilos, el hilo es la unidad de código más pequeña que se puede seleccionar para ejecución. La programación basada en multihilos permite escribir programas más eficientes que optimizan los recursos de la Unidad Central de Proceso (CPU), al reducir al mínimo los tiempos de inactividad, permitiendo ejecutar varias operaciones en paralelo y manejando los eventos inmediatamente desde su aparición. Este es un factor muy importante a tener en cuenta en el manejo de entornos interactivos, como es el caso del trabajo en la red, en donde las velocidades de transmisión son mucho más lentas que los requeridos por la CPU en el procesamiento de esos datos, así como también durante el manejo del sistema de archivos, lectura y grabación, que son más lentos que las velocidades de la CPU en su proceso.

1.7 Conclusiones del capítulo.

Con los elementos expuestos en este capítulo se muestran los distintos servicios que generalmente brinda un módulo de BDH, los cuales varían de un sistema a otro, cada uno con sus propias características. También se describieron de forma general los SGBD, enfatizando específicamente en los SBDR y los SGBD embebidos, las ventajas y particularidades a tener en cuenta al seleccionarlos según las necesidades y algunos ejemplos de los SGBD más usados actualmente en estas dos vertientes. Quedó evidenciada además, la importancia que tiene la incorporación de extensiones en sistemas que están sometidos a constantes cambios y que presentan una gran dependencia estructural entre las partes que lo componen. Este capítulo deja sentada las bases para un mayor entendimiento de las características del sistema a desarrollar, las cuales serán expuestas en el siguiente capítulo.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

2.1 Introducción

En este capítulo se muestra una visión práctica del sistema a desarrollar. En el mismo se exponen algunas de las herramientas y tecnologías empleadas, así como los requisitos funcionales y no funcionales que rigen el desarrollo de la solución propuesta. Partiendo de esto, se determinan los casos de uso y se describen los procesos de las principales funcionalidades del sistema.

2.2 Propósito

Cada vez que aumentan los compromisos que involucran al Centro de Informática Industrial, cada cliente demanda un conjunto de requisitos acorde a las características de su entidad. Este factor dificulta la integración del módulo de BDH existente, de manera que al tratar de suplir todas las necesidades se pierden recursos y tiempo en mantenimiento, al proveer las funcionalidades que se solicitan. El empleo de plugins o complementos permitiría que cualquiera de las funcionalidades que se quieran introducir, sean implantadas de forma independiente, sin afectar la estructura interna de la aplicación. Además de esto, si se dotara a la aplicación de un SGBD embebido de grandes potencialidades se resolverían los problemas de dependencia, evitando la necesidad de administradores adicionales para la base de datos y eliminando un componente más al cual rediseñar ante un cambio de requerimientos. De forma general, un módulo con estas características se adaptaría de forma más natural a los cambios, sería más ligero y fácil de desplegar.

2.3 Soluciones técnicas.

Para darle cumplimiento al objetivo de este trabajo se pretende desarrollar un módulo de gestión y archivo de datos que permita la introducción de las funcionalidades, a través de la incorporación de extensiones, y que sea independiente de un SBDR para la gestión de datos, usando para ello un SGBD embebido.

2.4 Tecnologías y herramientas a utilizar para el desarrollo.

Para la selección de las tecnologías y herramientas a utilizar en el desarrollo se tuvieron en cuenta una serie de requisitos como: su aceptación a nivel internacional, que fueran de libre distribución y fueran

además, las más adecuadas a los requerimientos del sistema a desarrollar. A continuación se enumera y describe cada una.

2.4.1 UML como lenguaje de modelado.

Se decidió utilizar el Lenguaje Unificado de Modelado (UML) por ser el lenguaje de modelado de sistemas de software más conocido a nivel mundial. Es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema. Está compuesto por elementos gráficos que se combinan para conformar diagramas. UML no tiene propietario y está basado en el común acuerdo de la comunidad informática.

Es importante resaltar que UML es un "lenguaje" para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional), pero no especifica en sí mismo qué metodología o proceso usar.

2.4.2 RUP como metodología de desarrollo.

Una metodología es el conjunto ordenado de pasos a seguir para cumplir un objetivo. Dicho objetivo, en la ingeniería de software, es el desarrollo de software de alta calidad que cumpla con las necesidades del cliente dentro de un plan y un presupuesto predecible.

Para el desarrollo de la solución se seleccionó la metodología del Proceso Unificado Racional (RUP por sus siglas en inglés). Este proceso de ingeniería de software se basa en la modelación de sistemas informáticos usando la tecnología orientada a objetos, lo que provee un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización desarrolladora o cualquier proyecto de software. RUP se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo consta de cuatro fases: inicio, elaboración, construcción y transición, y concluye con una versión del sistema, que está lista para ser entregada a los clientes.

2.4.3 Visual Paradigm como herramienta CASE.

Se puede definir a las Herramientas CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software (12).

Dentro de las herramientas CASE más renombradas se decidió usar Visual Paradigm, la cual como herramienta UML profesional soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue, incluye además actividades como la gestión de proyectos y la estimación. Este software de modelado UML contribuye con la rápida construcción de aplicaciones de calidad mejores y a un menor coste, es fácil de usar y permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Incluye además importación desde Rational Rose, exportación/importación XMI, generador de informes, editor de figuras, integración con MS Visio, plugin, integración IDE con Visual Studio, IntelliJ IDEA, Eclipse, NetBeans y otros. Apoya un conjunto de lenguajes tanto en la generación del código como en la Ingeniería Inversa por ejemplo Java, C + +, CORBA IDL, PHP, XML Schema, Ada y Python.

2.4.4 C++ como lenguaje de programación.

El uso de C++ como lenguaje de programación en el módulo viene dado por su utilización en el desarrollo del resto de los módulos del SCADA. También por su robustez, eficiencia e increíble versatilidad, permite programar desde el software más simple a los programas más complicados, como incluso sistemas operativos. Tiene la ventaja de ser portable, lo que significa que un programa escrito en C++ se puede compilar en cualquier sistema operativo sin la necesidad de muchos cambios en el código fuente.

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el de extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, C++ es un lenguaje híbrido. Una de sus particularidades es la posibilidad de redefinir los operadores y de poder crear nuevos tipos que se comporten como tipos fundamentales. También posee varios paradigmas de programación, como la programación estructurada y la programación orientada a objetos, permitiendo además facilidades para la programación genérica.

2.4.5 Eclipse como IDE de desarrollo.

Un entorno de desarrollo integrado (IDE por sus siglas en inglés) es un programa compuesto por una serie de herramientas para un programador. Puede estar dedicada a un lenguaje de programación en específico o bien puede utilizarse para varios. Entre las herramientas más comunes que poseen los IDEs están: un editor de código, un compilador, un intérprete, un depurador y un constructor de interfaces gráficas de usuario.

En la selección del IDE para desarrollar influyeron varios requisitos: se necesitaba que este permitiera la programación en C++, que contara con herramientas integradas para la gestión de la configuración tanto del código como de la documentación, que presentara un buen completamiento de código, que brindara facilidades en la vinculación de bibliotecas y en la configuración del compilador. El Eclipse cumple con todas las características expuestas anteriormente, además de posibilitar la construcción de un software multiplataforma de manera relativamente sencilla.

2.4.6 Berkeley DB como SGBD Embebido.

En el capítulo anterior se dio una introducción a Berkeley DB como SGBD Embebido. Después de un estudio realizado de las tecnologías para la gestión de BD de forma integrada, se decidió incluirlo como parte de la solución propuesta, debido a que las características que presenta son las que mayores beneficios aportan al proceso de desarrollo.

Berkeley DB permite el desarrollo de soluciones personalizadas para la gestión de datos, sin los gastos indirectos asociados tradicionalmente a proyectos de este tipo. Proporciona además una colección de tecnologías de construcción en bloque de probada eficacia que se puede configurar para hacer frente a cualquier necesidad, ya sea desde una solución de almacenamiento local a servir de núcleo en la construcción de un sistema distribuido. Berkeley DB es una solución confiable con más de 15 años en producción, en productos que van desde teléfonos celulares hasta aplicaciones de comercio electrónico.

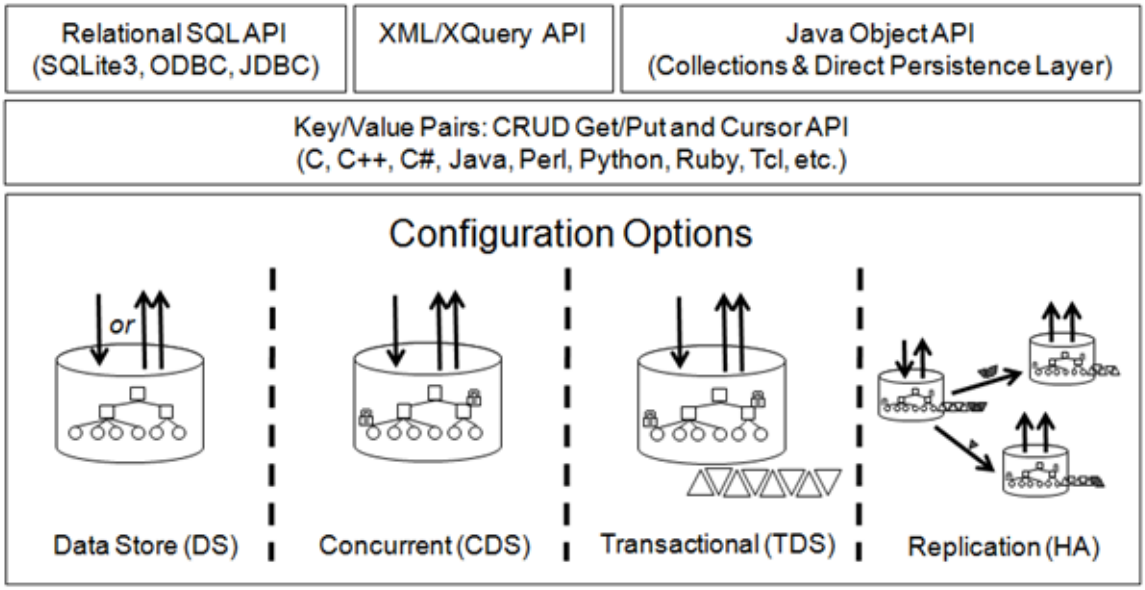


Figura 4: *Capas con las distintas API que provee Berkeley DB para el desarrollo.*

Algunos de los beneficios más importantes que se obtienen al usar Berkeley DB son:

- **Alto rendimiento:** En adición al hecho de ser un SGBD Embebido, su gran rendimiento radica en la flexibilidad que posee para personalizar la BD a los requerimientos del sistema que lo utilice.
- **Alta fiabilidad y disponibilidad:** Su completa semántica transaccional garantiza la integridad de los datos, recuperación ante fallos y replicación.
- **Cero administraciones:** No requiere administradores adicionales de BD, debido a que toda la administración se realiza vía API, ocultándose al usuario final.
- **Múltiples modelos de datos:** Se manejan distintos modelos de datos a través de las interfaces: SQL, clave/valor, Java.
- **SQL API:** Basada en la conocida API SQLite3, Berkeley DB proporciona una interfaz para el manejo de bases de datos SQL y una herramienta para su tratamiento a través de la línea de comandos.

- **Soporte para los estándares ODBC y JDBC:** Posibilita el acceso a las BD en disco usando manejadores para estos estándares.

Como muestra de su gran desempeño es usado por empresas de renombrado prestigio como: Airbus, Amazon, AOL, Cisco Systems, eBay, EMC, Google, Hitachi, HP, Motorola, Nortel, RSA Security, Sun Microsystems, TIBCO y VERITAS.

2.4.7 Biblioteca Boost de C++ para el manejo de hilos y sistema de archivos.

Boost es un conjunto de bibliotecas de código abierto y revisión por pares preparadas para extender las capacidades del lenguaje de programación C++. Su licencia permite que sea utilizada en cualquier tipo de proyectos, ya sean comerciales o no. Varios fundadores de Boost pertenecen al Comité ISO de Estándares C++, aunque posteriormente han recibido aportes de múltiples autores.

Lo primero a destacar de estas bibliotecas es su alta calidad técnica, su diseño e implementación permiten que sea utilizada en un amplio espectro de aplicaciones y plataformas, abarcando desde librerías de propósito general hasta abstracciones del sistema operativo. El proceso de selección para que una biblioteca sea admitida es público y bastante estricto, garantizando que el mero hecho de aparecer en la colección es un marchamo de calidad y nivel técnico.

2.4.8 Debian como sistema operativo.

Debido a que la mayoría de los productos surgidos en el Centro se han desarrollado sobre el sistema operativo GNU Linux en su distribución Debian, por todas las ventajas que ofrece, se decidió mantenerlo como plataforma para la elaboración del módulo.

Como casi todas las distribuciones de GNU Linux, Debian es multitarea, multiusuario, multiplataforma, multiprocesador y en las plataformas Intel se ejecuta en modo protegido. Protege la memoria para que un programa no pueda hacer caer el resto del sistema; carga solo las partes de un programa que se usan, lo que proporciona una mayor eficiencia; comparte la memoria entre programas aumentando la velocidad y disminuyendo el uso de memoria; utiliza toda la memoria libre para caché; permite usar bibliotecas enlazadas tanto estática como dinámicamente, además de ser un sistema operativo libre por el que no se debe pagar una patente y se distribuye con su código fuente, permitiendo adaptar el sistema a las necesidades de cada desarrollador.

2.5 Modelo de Dominio.

En la siguiente figura se muestra el modelo de dominio, en él se representan de manera visual los principales conceptos y relaciones que se manejan dentro del dominio del sistema a desarrollar.

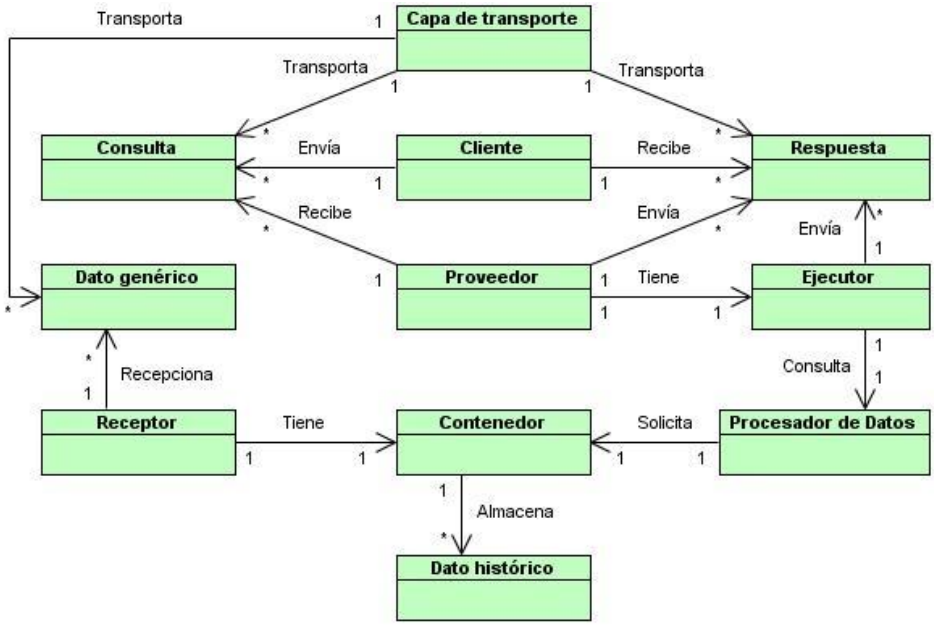


Figura 5: *Modelo de dominio.*

2.5.1 Clases Conceptuales o Glosario de términos del dominio.

- **Capa de transporte:** Módulo encargado de la transferencia de datos (Middleware).
- **Consulta:** Solicitud de datos de los históricos.
- **Respuesta:** Resultado de la ejecución de una consulta o mensaje de error.
- **Cliente:** Encargado de llevar a cabo una solicitud de consulta y luego procesar la respuesta recibida.
- **Proveedor:** Gestiona las solicitudes de consultas a la BD. Genera una respuesta en caso de algún error o que el servidor este ocupado.

- **Ejecutor:** Pone en ejecución una consulta y devuelve la(s) respuesta(s) a través de la capa de transporte.
- **Procesador de datos:** Contiene la lógica de almacenamiento y acceso a los datos históricos que persisten en la BD.
- **Receptor:** Encargado de recibir los datos transmitidos a través de la capa de transporte.
- **Dato genérico:** Representa una estructura genérica de datos.
- **Contenedor:** Convierte los datos genéricos recibidos y los transforma a datos históricos para luego almacenarlos temporalmente hasta ser solicitados por el procesador de datos.
- **Dato histórico:** Representa una estructura de datos definida para su almacenamiento histórico.

2.6 Descripción del sistema propuesto.

El sistema propuesto se estructura en dos partes fundamentales: los plugins o componentes y el servidor de BD de Históricos. Los plugins constituyen la base funcional del módulo, estos encapsulan las principales funcionalidades que estarían sujetas a modificaciones ante un cambio de requisitos, estas son:

- **Recepción de información:** Implementación de mecanismos para la recepción de datos desde un sistema determinado.
- **Suministro de información:** Implementación de mecanismos para proporcionar los datos a través de determinado tipo de consulta, como puede ser la solicitud de variables históricas vista en el epígrafe 1.3.5.
- **Procesamiento de información:** Implementación de los diferentes métodos para el procesamiento de los datos a almacenar, como puede ser la aplicación de métodos de filtrado, compresión, reciclaje de los datos al llegar a un determinado límite de tiempo, entre otras.

Encapsular todas estas funcionalidades en plugins permite independizar cada solución en un componente por separado que funciona sin afectar el desempeño del resto. Cada plugin puede ser desarrollado y

modificado para adaptarse a los requisitos de determinada entidad sin afectar la estructura del servidor de BD de Históricos. Además de esto, permiten que con ninguna o pocas modificaciones puedan ser reutilizados en distintos escenarios que posean características comunes o puedan ser retirados en el momento que se requiera, todo esto sin alterar el funcionamiento general del sistema.

Por su parte el Servidor de BD de Históricos es el encargado de ejecutar las tareas relacionadas con la gestión de los plugins, como cargarlos, poner en ejecución el servicio que implementan, detener la ejecución del servicio, eliminarlos, entre otras. También es el controlador en el acceso a datos, proporcionando el acceso a todas las tablas de BD que utiliza cada plugin para guardar su información.

En la figura 6 se muestra una visión general de la estructura que tendría el módulo de Gestión y Archivo de Datos.

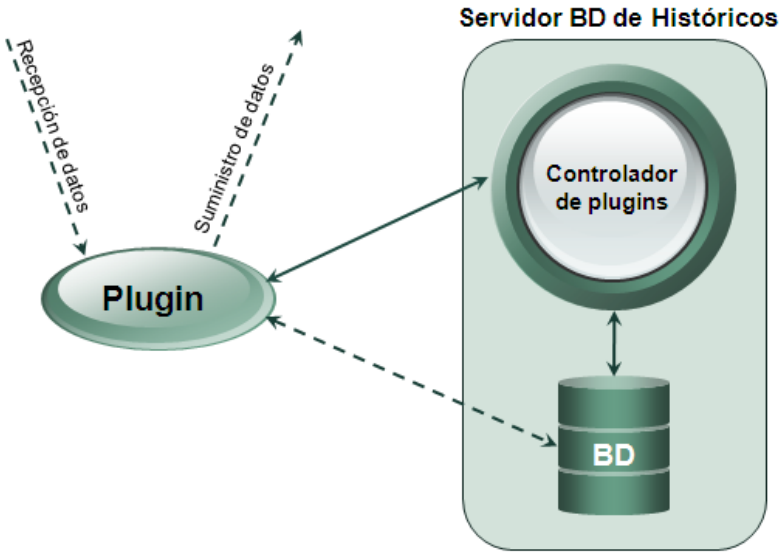


Figura 6: *Estructura general del módulo de Gestión y Archivo de Datos.*

Los plugins se construyen como una biblioteca dinámica, usando para ello el mismo lenguaje C++. Estos deben implementar una interfaz que permita la interacción desde el sistema (Servidor de BD de Históricos), encapsulando funcionalidades para poner en funcionamiento el servicio específico que brindan, la posibilidad además de detenerlo en el momento que se requiera y otras acciones que forman parte de los requisitos fundamentales que debe cumplir el sistema.

Cada plugin como parte de su inicialización implementa una rutina que realiza una solicitud al servidor con el número de tablas de BD que necesita, sus nombres y los tipos de datos que guardan cada una; si no hay ningún tipo de error en dichas solicitudes, el servidor le entrega a cada plugin un grupo de acceso a las tablas solicitadas, las cuales si no existían físicamente son creadas. Luego de tener sus grupos de acceso, los plugins solo podrán insertar, eliminar y consultar datos, pero nunca eliminar físicamente las tablas; esa responsabilidad la tiene el servidor, el cual la usa solamente cuando un plugin es eliminado completamente.

2.6.1 Requisitos funcionales.

Los requerimientos funcionales describen lo que el sistema debe hacer: son todas las condiciones y capacidades que debe cumplir el software, o producto en general, para que las peticiones del cliente queden satisfechas.

Como pudo apreciarse al inicio del epígrafe 2.6, el desarrollo de cada plugin responde a las necesidades específicas del escenario en el que se quiere aplicar, por tanto, el servidor de BD de históricos solo se limita a interactuar con los mismos, abstrayéndose de cómo implementan internamente sus funcionalidades. Debido a esto, el mayor peso de los requisitos funcionales que el sistema debe cumplir se enfoca directamente en la interacción del mismo con los plugins desarrollados.

Dentro de los requisitos funcionales presentes, el sistema debe permitir:

RF1: Cargar los plugins.

RF2: Inicializar los plugins.

RF3: Ejecutar el servicio de los plugins.

RF4: Detener la ejecución de servicio de los plugins.

RF5: Eliminar los plugins.

RF6: Listar los plugins.

RF7: Consultar la versión de los plugins.

2.6.2 Requisitos no funcionales.

Los requerimientos no funcionales son definidos como propiedades o cualidades que el producto debe tener, son aspectos importantes que este debe cumplir para lograr un aprovechamiento óptimo de las funcionalidades del sistema, teniendo en cuenta el entorno en el que será utilizado. A continuación se enuncian, separados en categorías, los diferentes requisitos no funcionales.

Requisitos de Software

RNF 1: Sistema Operativo Debian GNU/Linux 5.0 Lenny, Kernel 2.6.26-1-686.

Requisitos de Hardware

RNF 2: Memoria RAM 256 MB o superior, microprocesador: 1.0 GHz o superior, tarjeta de red.

Restricciones en el diseño y la implementación

RNF 3: Rendimiento y alta disponibilidad: El sistema debe poseer características de rendimiento y alta disponibilidad de forma que se garantice su operatividad de forma segura, efectiva y confiable.

RNF 4: Emplear el paradigma de programación orientado a objetos y C++ como lenguaje de programación.

Requisitos de Seguridad

RNF 5: Confidencialidad: La información manejada por el sistema deberá estar protegida de acceso no autorizado y divulgación.

RNF 6: Integridad: El sistema debe permitir la conservación de los plugins, además de ser capaz de mantener el estado y la calidad de los datos almacenados por cada uno en la BD.

Requisitos de Usabilidad

RNF 7: El sistema debe poseer un procedimiento sencillo de puesta en marcha y uso, garantizando al operador un fácil e intuitivo acceso al mismo.

Requisitos de Soporte

RNF 8: El sistema debe ser de fácil instalación, configuración y puesta en marcha.

RNF 9: Licencia bajo código abierto: El sistema debe cumplir con los lineamientos necesarios para la producción de software libre y la comunidad de desarrollo y soporte.

2.6.3 Actores del Sistema.

Actor	Descripción
Operador	Es la persona encargada de la puesta en marcha del sistema, así como la supervisión del mismo.

Tabla 1: *Descripción de los actores.*

2.6.4 Casos de Uso del Sistema.

Cada caso de uso del sistema puede ser catalogado como crítico, secundario, auxiliar u opcional, en correspondencia a la importancia que estos tengan dentro del sistema y atendiendo a las peticiones del cliente. A continuación se relacionan los casos de uso correspondientes al presente trabajo de diploma y sus respectivas clasificaciones:

Críticos	Secundarios	Auxiliares
Registrar plugin Ejecutar servicio de plugin Detener servicio de plugin	Listar plugins Consultar versión del plugin	Eliminar plugin

Tabla 2: *Clasificación de los Casos de Uso del Sistema.*

2.6.5 Diagrama de Casos de Uso del Sistema.

El modelado de Casos de Uso se realiza con el objetivo de modelar de una forma simple y efectiva los requisitos del sistema desde el punto de vista del usuario. El diagrama de Casos de Uso constituye una visión general que se ha identificado para satisfacer los requerimientos funcionales del sistema.

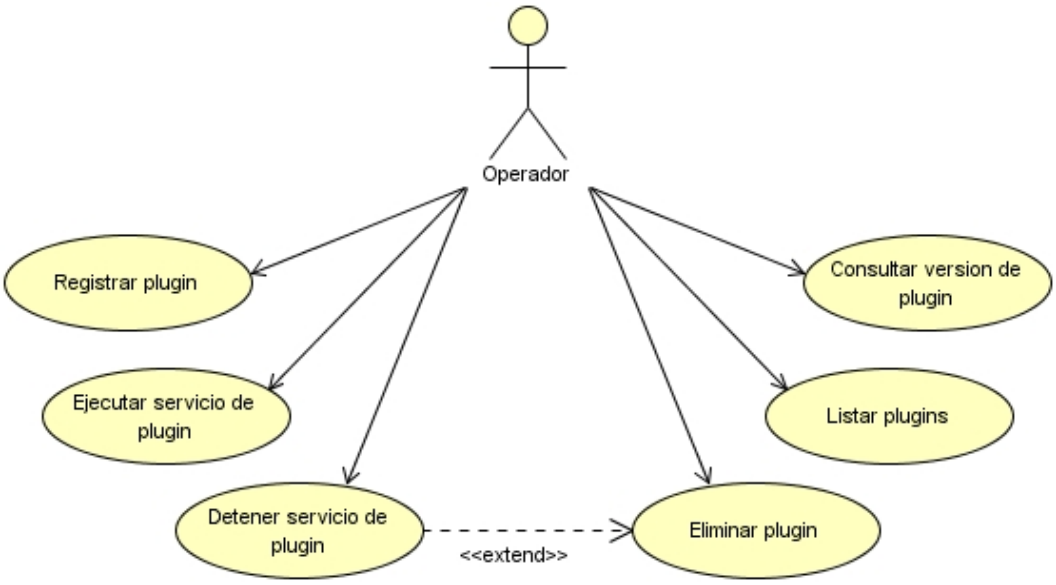


Figura 7: *Diagrama de Caso de Usos del Sistema.*

2.6.6 Descripción de los Casos de Uso del Sistema.

La descripción detallada de los Casos de Uso muestra la expansión que permite comprender los procesos que se encuentran asociados a cada uno de ellos. A continuación se muestran las expansiones para los Casos de Uso definidos anteriormente.

2.6.6.1 Registrar plugin.

Nombre del CU	Registrar plugin.
Objetivo	El actor tiene como objetivo registrar un plugin determinado.
Actores	Operador (Inicia)
Resumen	El caso de uso se inicia cuando el Operador desea registrar un plugin.
Complejidad	Alta.
Prioridad	Crítico.
Precondiciones	La biblioteca dinámica que contiene al plugin debe encontrarse en el directorio de plugins definido en la configuración del sistema.
Postcondiciones	El plugin debió haber quedado registrado e inicializado en el sistema.
Flujo de eventos	
Flujo básico	
Actor	**Sistema**
1. Envía un comando con el nombre de la	2. Verifica la sintaxis del comando. Si es

biblioteca dinámica que contiene al plugin a ser registrado.		incorrecta (Ver flujo alterno 1).
		3. Verifica si no hay un plugin registrado con el mismo nombre que el plugin de la biblioteca. Si ya existe uno (Ver flujo alterno 2).
		4. Registrar plugin.
		5. Inicializar plugin.
		6. Muestra un mensaje de confirmación.
		7. Termina el caso de uso.
Flujos alternos		
Nº1. La sintaxis del comando es incorrecta.		
Actor		**Sistema**
		1. Muestra un mensaje de error. 2. Retorna al paso 7.
Nº2. Existe un plugin registrado con el mismo nombre.		
Actor		**Sistema**
		1. Muestra un mensaje de error. 2. Retorna al paso 7.
Relaciones	**CU Incluidos**	No tiene.
	CU Extendidos	No tiene.
Requisitos funcionales	RF 1, RF 2	
Prototipo interfaz	Ver anexo 6.	

Tabla 3: *Descripción del CUS "Registrar plugin".*

2.6.6.2 Ejecutar servicio de plugin.

Nombre del CU	Ejecutar servicio de plugin.
Objetivo	El actor tiene como objetivo ejecutar el servicio de un plugin determinado.
Actores	Operador (Inicia)
Resumen	El caso de uso se inicia cuando el Operador desea poner en ejecución el servicio que brinda un plugin específico.
Complejidad	Alta.
Prioridad	Crítico.
Precondiciones	El plugin debe haber sido registrado e inicializado por el sistema.
Postcondiciones	El plugin debió haber quedado en estado de ejecución.
Flujo de eventos	

Flujo básico		
Actor		**Sistema**
1. Envía un comando con el nombre del plugin del cual quiere ejecutar su servicio.		2. Verifica la sintaxis del comando. Si es incorrecta (Ver flujo alterno 1).
		3. Verifica si el plugin ya se encuentra en estado de ejecución. Si ya lo está (Ver flujo alterno 2).
		4. Pone en ejecución el servicio del plugin.
		5. Muestra un mensaje de confirmación.
		6. Termina el caso de uso.
Flujos alternos		
Nº1. La sintaxis del comando es incorrecta.		
Actor		**Sistema**
		1. Muestra un mensaje de error.
		2. Retorna al paso 6.
Nº2. El plugin ya se encuentra en ejecución.		
Actor		**Sistema**
		1. Muestra un mensaje de información.
		2. Retorna al paso 6.
Relaciones	**CU Incluidos**	No tiene.
	CU Extendidos	No tiene.
Requisitos funcionales	RF 3	
Prototipo interfaz	Ver anexo 9.	

Tabla 4: *Descripción del CUS "Ejecutar servicio de plugin".*

2.6.6.3 Detener servicio de plugin.

Nombre del CU	Detener servicio de plugin.
Objetivo	El actor tiene como objetivo detener el servicio de un plugin determinado.
Actores	Operador (Inicia)
Resumen	El caso de uso se inicia cuando el Operador desea detener la ejecución del servicio de un plugin específico.
Complejidad	Alta.
Prioridad	Crítico.
Precondiciones	El plugin debe haber sido registrado e inicializado por el sistema.
Postcondiciones	El plugin debió haber detenido la ejecución del su servicio.

Flujo de eventos		
Flujo básico		
Actor		**Sistema**
1. Envía un comando con el nombre del plugin del cual quiere detener su servicio.		2. Verifica la sintaxis del comando. Si es incorrecta (Ver flujo alterno 1).
		3. Verifica si el plugin ya se encuentra con el servicio detenido. Si ya lo está (Ver flujo alterno 2).
		4. Detiene ejecución el servicio del plugin.
		5. Muestra un mensaje de confirmación.
		6. Termina el caso de uso.
Flujos alternos		
Nº1. La sintaxis del comando es incorrecta.		
Actor		**Sistema**
		1. Muestra un mensaje de error.
		2. Retorna al paso 6.
Nº2. El plugin ya se encuentra con el servicio detenido.		
Actor		**Sistema**
		1. Muestra un mensaje de información.
		2. Retorna al paso 6.
Relaciones	**CU Incluidos**	No tiene.
	CU Extendidos	No tiene.
Requisitos funcionales	RF 4	
Prototipo interfaz	Ver anexo 10.	

Tabla 5: *Descripción del CUS "Detener servicio de plugin".*

2.6.6.4 Eliminar plugin.

Nombre del CU	Eliminar plugin.
Objetivo	El actor tiene como objetivo eliminar un plugin determinado.
Actores	Operador (Inicia)
Resumen	El caso de uso se inicia cuando el Operador desea eliminar un plugin.
Complejidad	Alta.
Prioridad	Auxiliar.
Precondiciones	El plugin debe haber sido registrado e inicializado por el sistema.
Postcondiciones	El plugin debió haber quedado eliminado, así como toda su información

	generada como: tablas de BD, directorio de configuración y directorio cache.	
Flujo de eventos		
Flujo básico		
Actor	**Sistema**	
1. Envía un comando con el nombre del plugin que se quiere sea eliminado.	2. Verifica la sintaxis del comando. Si es incorrecta (Ver flujo alterno 1).	
	3. Verifica que el plugin a eliminar no se encuentra en estado de ejecución. Si se encuentra en ejecución (Ver flujo alterno 2).	
	4. Eliminar plugin.	
	5. Muestra un mensaje de confirmación.	
	6. Termina el caso de uso.	
Flujos alternos		
Nº1. La sintaxis del comando es incorrecta.		
Actor	**Sistema**	
	1. Muestra un mensaje de error. 2. Retorna al paso 6.	
Nº2. El plugin a eliminar se encuentra en estado de ejecución.		
Actor	**Sistema**	
	1. Ejecuta el CU “Detener servicio de plugin”. 2. Retorna al paso 4.	
Relaciones	**CU Incluidos**	No tiene.
	CU Extendidos	Detener servicio de plugin.
Requisitos funcionales	RF 5	
Prototipo interfaz	Ver anexo 8.	

Tabla 6: *Descripción del CUS “Eliminar plugin”.*

2.6.6.5 Listar plugins.

Nombre del CU	Listar plugins.
Objetivo	El actor tiene como objetivo listar los plugins que se encuentran registrados en el sistema.
Actores	Operador (Inicia)
Resumen	El caso de uso se inicia cuando el Operador desea listar los plugins que se encuentran registrados en el sistema, así como sus estados.
Complejidad	Baja.

Prioridad	Secundario.	
Precondiciones	No tiene.	
Postcondiciones	El sistema debió haber mostrado el listado de los plugins registrados.	
Flujo de eventos		
Flujo básico		
Actor	**Sistema**	
1. Envía un comando con la petición de listar los plugins registrados.	2. Verifica la sintaxis del comando. Si es incorrecta (Ver flujo alterno 1).	
	3. Verifica si existen plugins registrados. Si no hay plugins registrados (Ver flujo alterno 2).	
	4. Listar plugins.	
	5. Termina el caso de uso.	
Flujos alternos		
Nº1. La sintaxis del comando es incorrecta.		
Actor	**Sistema**	
	1. Muestra un mensaje de error. 2. Retorna al paso 5.	
Nº2. No hay plugins registrados.		
Actor	**Sistema**	
	1. Muestra un mensaje de información. 2. Retorna al paso 5.	
Relaciones	**CU Incluidos**	No tiene.
	CU Extendidos	No tiene.
Requisitos funcionales	RF 6	
Prototipo interfaz	Ver anexo 7.	

Tabla 7: *Descripción del CUS "Listar plugins".*

2.6.6.6 Consultar versión de plugin.

Nombre del CU	Consultar versión del plugin.
Objetivo	El actor tiene como objetivo consultar la versión de un plugin determinado.
Actores	Operador (Inicia)
Resumen	El caso de uso se inicia cuando el Operador desea consultar la versión de un plugin específico, que incluye el nombre del proveedor y la descripción.
Complejidad	Baja.
Prioridad	Secundario.

Precondiciones	El plugin debe haber sido registrado e inicializado por el sistema.	
Postcondiciones	El sistema debió haber mostrado la versión del plugin.	
Flujo de eventos		
Flujo básico		
Actor	**Sistema**	
1. Envía un comando con el nombre del plugin del cual quiere consultar su versión.	2. Verifica la sintaxis del comando. Si es incorrecta (Ver flujo alterno 1).	
	3. Muestra la versión del plugin, el nombre del proveedor y la descripción.	
	4. Termina el caso de uso.	
Flujos alternos		
Nº1. La sintaxis del comando es incorrecta.		
Actor	**Sistema**	
	1. Muestra un mensaje de error. 2. Retorna al paso 4.	
Relaciones	**CU Incluidos**	No tiene.
	CU Extendidos	No tiene.
Requisitos funcionales	RF 7	
Prototipo interfaz	Ver anexo 12.	

Tabla 8: *Descripción del CUS "Consultar versión de plugin".*

2.7 Conclusiones del capítulo.

En este capítulo han sido sentadas las bases técnicas por las que se regirá el desarrollo del sistema propuesto. Partiendo de esto, en el siguiente capítulo se diseñará e implementará una estructura de clases en concordancia con lo definido previamente en este capítulo y de esta forma darle cumplimiento a los objetivos del presente trabajo.

CAPÍTULO 3: DISEÑO, IMPLEMENTACIÓN Y PRUEBAS

3.1 Introducción

En este capítulo se procede a la realización del diseño e implementación del sistema. En él se exponen los diagramas de clases de diseño separados por paquetes, con la descripción de las clases que conforman a cada uno. Como parte de la transición de la fase de diseño a la de implementación, se presenta el diagrama de componentes del sistema. Se dedica un epígrafe a explicar el proceso de desarrollo de plugins y por último se procede a la realización de pruebas al sistema, para validar su correcto funcionamiento.

3.2 Diseño del sistema.

Como se expuso en el capítulo anterior el enfoque principal del sistema es la gestión de plugins o componentes que manejan datos históricos. En su diseño el sistema se divide en un grupo de paquetes que conforman su arquitectura, cada uno contiene un grupo de clases que se relacionan entre sí, de acuerdo con las funciones que realizan. La figura 7 muestra estos paquetes, así como la relación que existe entre ellos a través de las clases que los integran.

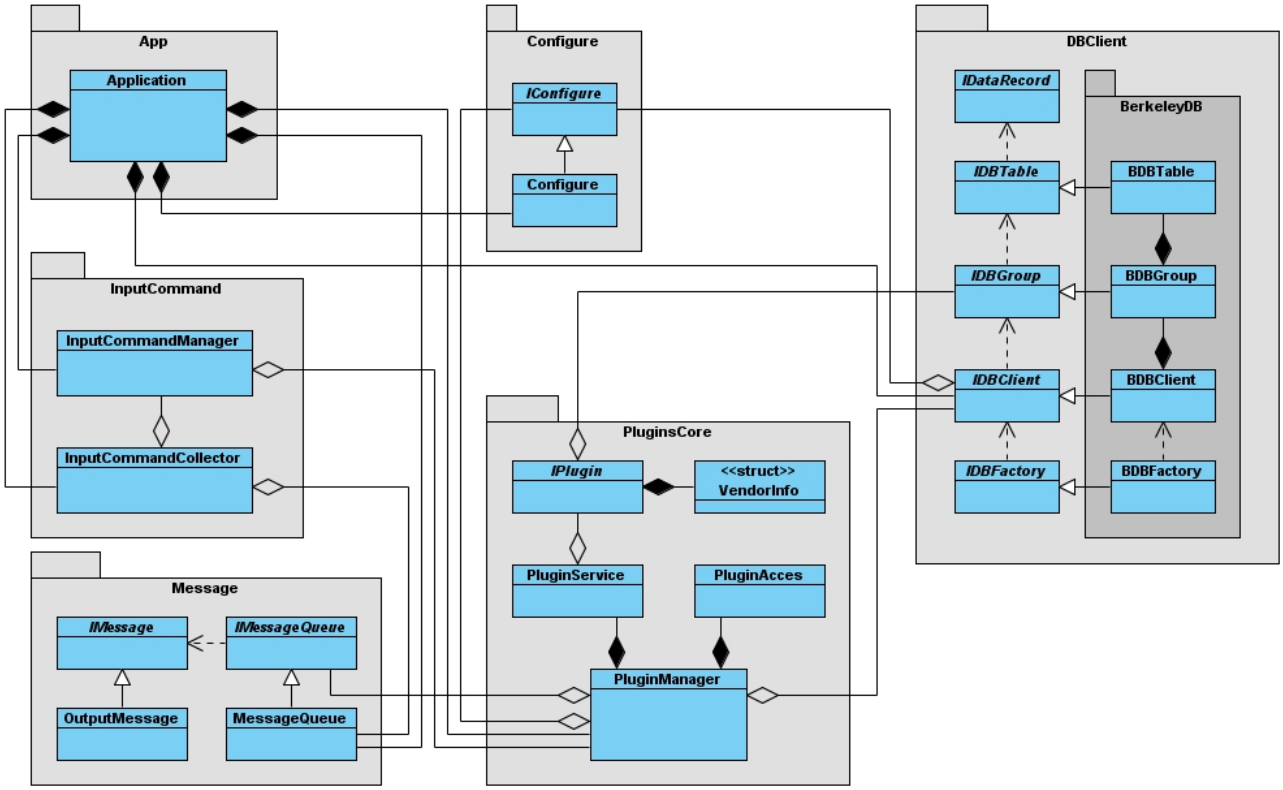


Figura 8: *Paquetes del sistema.*

A continuación se expone en detalle cada paquete y las clases que contiene cada uno. Para mayor claridad se agregaron las clases de otros paquetes que poseen relación con las clases del paquete descrito. Estas últimas se colorearon de azul, mientras que las clases externas se colorearon de amarillo y se le omitieron los atributos y funciones miembros.

3.2.1 Paquete App.

El paquete App es un paquete simbólico cuyo objetivo es solamente el de encapsular la clase Application.

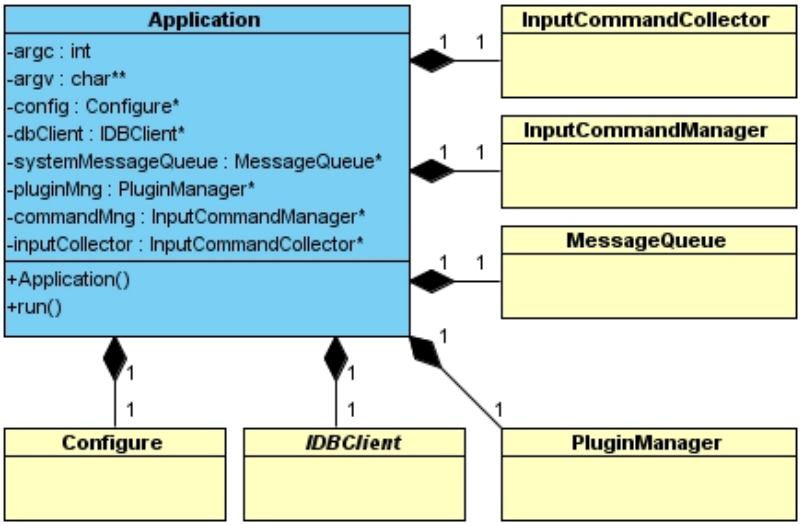


Figura 9: *Diagrama de Clases del paquete App.*

Descripción de las clases del paquete App	
Nombre de la clase	**Descripción**
Application	Esta es la clase principal de la aplicación. Es la encargada de la puesta en marcha del sistema, contiene todas las instancias de las clases principales de cada paquete y es responsable de su interacción y correcto funcionamiento.

Tabla 9: *Descripción de las clases del paquete App.*

3.2.2 Paquete Configure.

El paquete Configure contiene las clases relacionadas con el manejo de toda la configuración del sistema. La información de configuración brindada por el paquete a través de sus clases radica en un conjunto de direcciones donde el sistema localiza sus datos, como: directorio donde se encuentran los plugins, directorio donde se guardan las BD, directorio de configuración, entre otras. Todas estas direcciones se editan en un archivo de configuración, cuya dirección es pasada como argumento al iniciar la aplicación desde un terminal o consola.

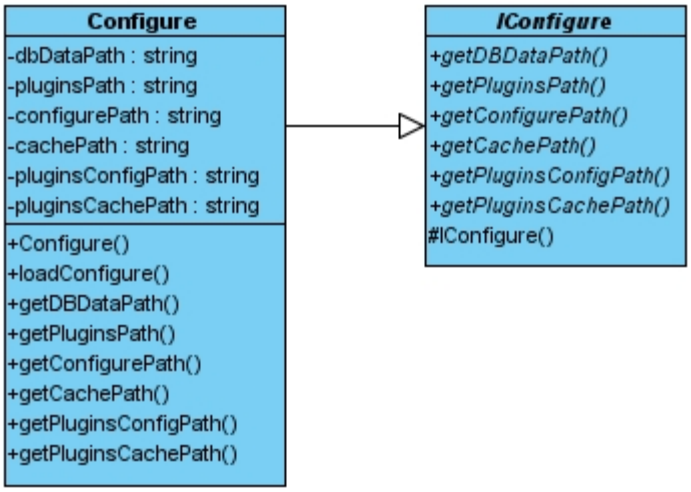


Figura 10: *Diagrama de Clases del paquete Configure.*

Descripción de las clases del paquete Configure	
Nombre de la clase	**Descripción**
IConfigure	Esta clase funciona como interfaz a implementar por la clase responsable de manejar la configuración del sistema. Su utilidad radica en servir como apuntador a una clase de configuración concreta, para permitir el acceso de consulta a las clases del sistema que la requieran.
Configure	Es la clase concreta que implementa la interfaz IConfigure. Es la encargada de cargar desde un archivo de configuración toda la información necesaria para la puesta en marcha del sistema.

Tabla 10: *Descripción de las clases del paquete Configure.*

3.2.3 Paquete InputCommand.

El paquete InputCommand implementa toda la lógica de interacción entre el operador y el sistema, usando para ello una arquitectura de N-Capas o *n-tier*. En la figura 10 se muestra la distribución de las clases del paquete en la aplicación de este patrón arquitectónico.

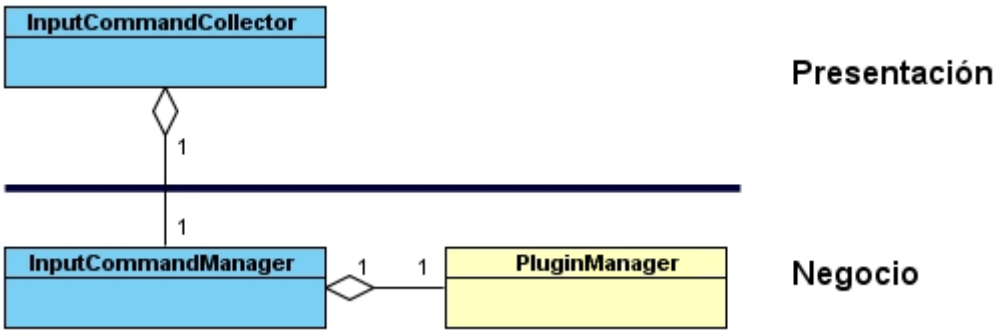


Figura 11: *Arquitectura en capas del paquete InputCommand.*

La clase InputCommandCollector se mantiene en ejecución en un hilo independiente al programa principal. Esta se queda en espera de la introducción de comandos por parte del usuario desde un terminal o consola, conformando así la capa de presentación. Luego delega la responsabilidad de la validación y ejecución de los comandos a la clase InputCommandManager, la cual utiliza las funcionalidades de la clase manejadora de plugins (PluginsCore) como parte de la ejecución de los comandos relacionados con la gestión de los plugins; estas dos clases como puede observarse en el diagrama conforman la capa de negocio.

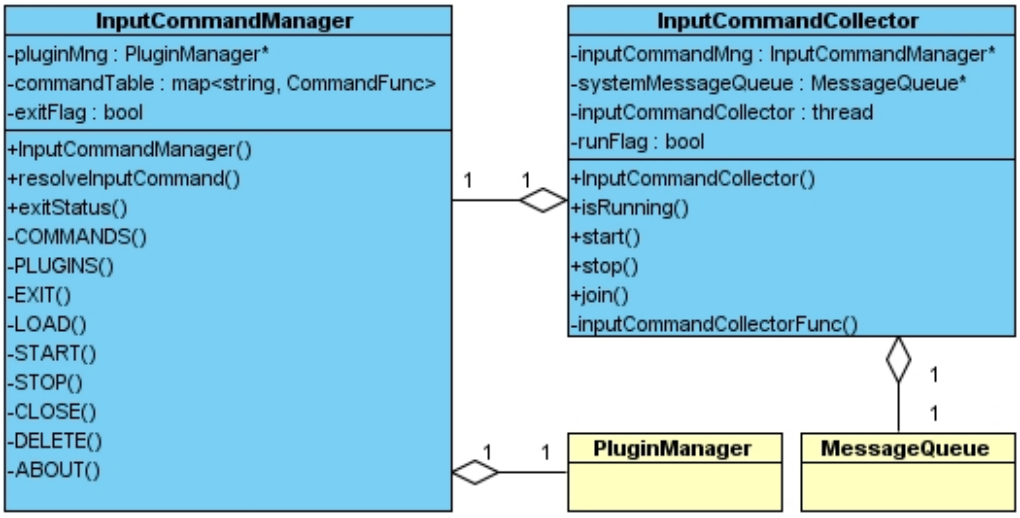


Figura 12: *Diagrama de Clases del paquete InputCommand.*

Descripción de las clases del paquete InputCommand	
Nombre de la clase	**Descripción**
InputCommandCollector	Esta clase es la encargada de controlar todos los procesos de entrada-salida del sistema hacia el terminal o consola. Mantiene activo un hilo de ejecución encargado de recepcionar los comandos entrados por el usuario para interactuar con el sistema, mostrando además todos los mensajes generados por el mismo.
InputCommandManager	Es la clase encargada de validar y resolver los comandos recibidos desde InputCommandCollector. Para la resolución de los comandos relacionados con la gestión de plugins, la clase se apoya en el uso de la clase controladora PluginManager.

Tabla 11: *Descripción de las clases del paquete InputCommand.*

3.2.4 Paquete Message.

Este paquete contiene todas las clases relacionadas con el manejo de mensajes dentro del sistema.

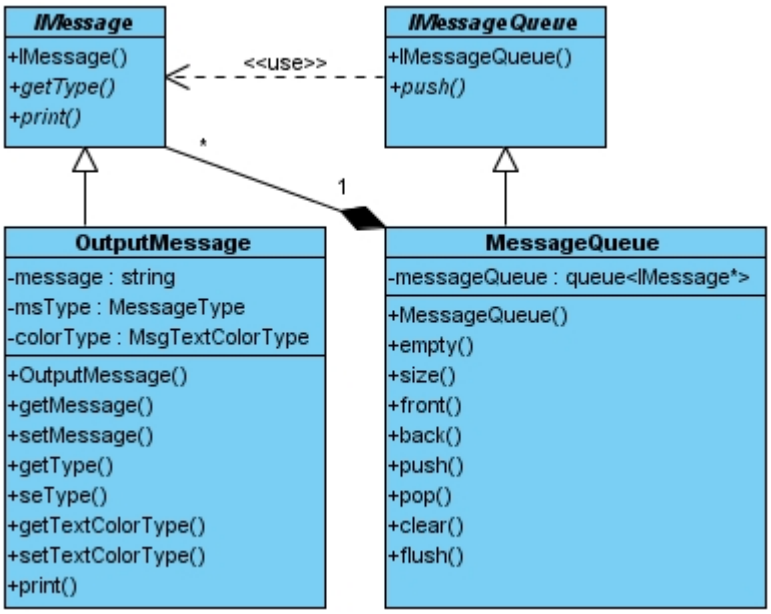


Figura 13: *Diagrama de Clases del paquete Message.*

Descripción de las clases del paquete Message	
Nombre de la clase	**Descripción**
IMessage	Clase que sirve de interfaz a implementar por la clase OutputMessage.
OutputMessage	Implementa la interfaz IMessage. Esta clase encapsula el concepto de mensaje de texto, el cual entre otras funcionalidades, posee la particularidad de poder imprimir su contenido por el terminal o consola.
IMessageQueue	Clase que sirve de interfaz a implementar por la clase MessageQueue.
MessageQueue	Implementa la interfaz IMessageQueue. Esta clase encapsula el concepto de cola de mensajes. Una instancia de su tipo perteneciente a la clase Application es utilizada por la clase InputCommandCollector para el manejo de mensajes generados desde la clase PluginManager.

Tabla 12: *Descripción de las clases del paquete Message.*

3.2.5 Paquete PluginsCore.

Como su nombre lo indica este paquete contiene todas las clases relacionadas con la manipulación de los plugins en el sistema.

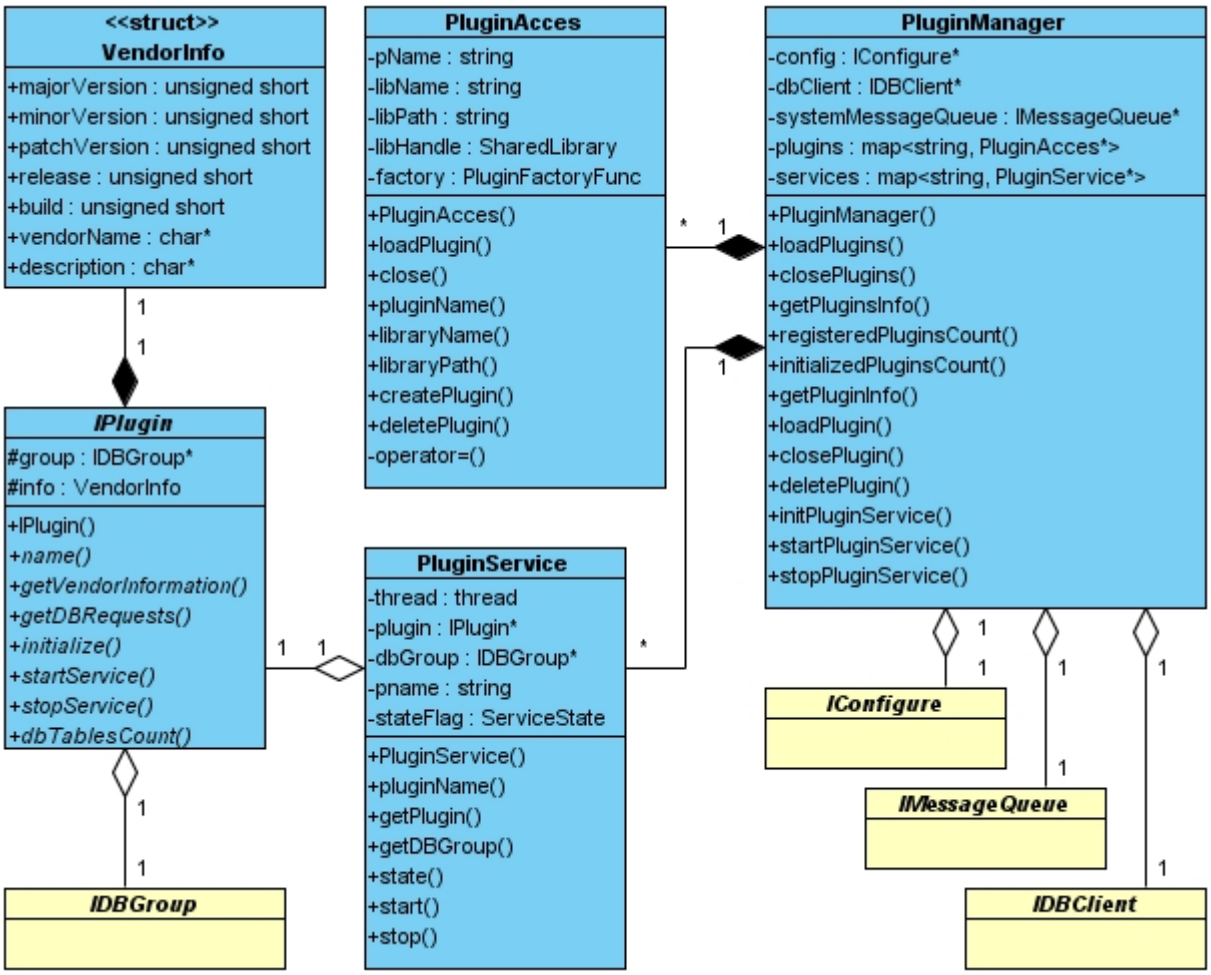


Figura 14: *Diagrama de Clases del paquete PluginsCore.*

Descripción de las clases del paquete PluginsCore	
Nombre de la clase	**Descripción**
VendorInfo	Estructura que contiene cada plugin con los datos de su proveedor.
IPlugin	Es una de las interfaces de mayor importancia en todo el diseño del sistema, pues todos los plugins desarrollados deben implementarla, permitiendo al sistema interactuar con las particularidades que exporta cada uno.

PluginAcces	Clase que guarda el registro de un plugin determinado al ser cargado en el sistema. Contiene además la fábrica que exporta el plugin para la creación de instancias del mismo.
PluginService	Encapsula la instancia de un determinado plugin registrado previamente en el sistema. Esta clase permite ejecutar o detener la ejecución de servicio del plugin que acoge, a través de las funcionalidades que este exporta. En el caso de la ejecución de servicio, la clase ejecuta en un hilo de ejecución independiente la implementación de la función del plugin dedicada a esta actividad.
PluginManager	Es la clase controladora de todos los plugins registrados en el sistema. Posee un grupo de funcionalidades para manejo de plugins como: cargarlos, inicializarlos, cerrarlos, eliminarlos, entre otras.

Tabla 13: *Descripción de las clases del paquete PluginsCore.*

3.2.6 Paquete DBClient.

El paquete DBClient contiene todas las clases relacionadas con la gestión de BD dentro del sistema. El conjunto de interfaces que posee permite aplicar el patrón de diseño *Abstract Factory* (Fábrica abstracta). Este patrón permite trabajar con objetos de distintas familias, de manera que las familias no se mezclen entre sí, haciendo transparente el tipo de familia concreta que se esté usando; en este caso las familias de objetos estarían constituidas por el tipo SGBD usado. Entre las ventajas que ofrece la aplicación de este patrón está que el sistema puede hacer uso de las BD indistintamente a través de estas interfaces, sin tener que conocer el tipo de SGBD usado, esto permite que ante un futuro cambio de gestor, la arquitectura general del sistema no se vea afectada. También da la posibilidad de incorporar implementaciones adicionales de las interfaces usando nuevos tipos de SGBD.

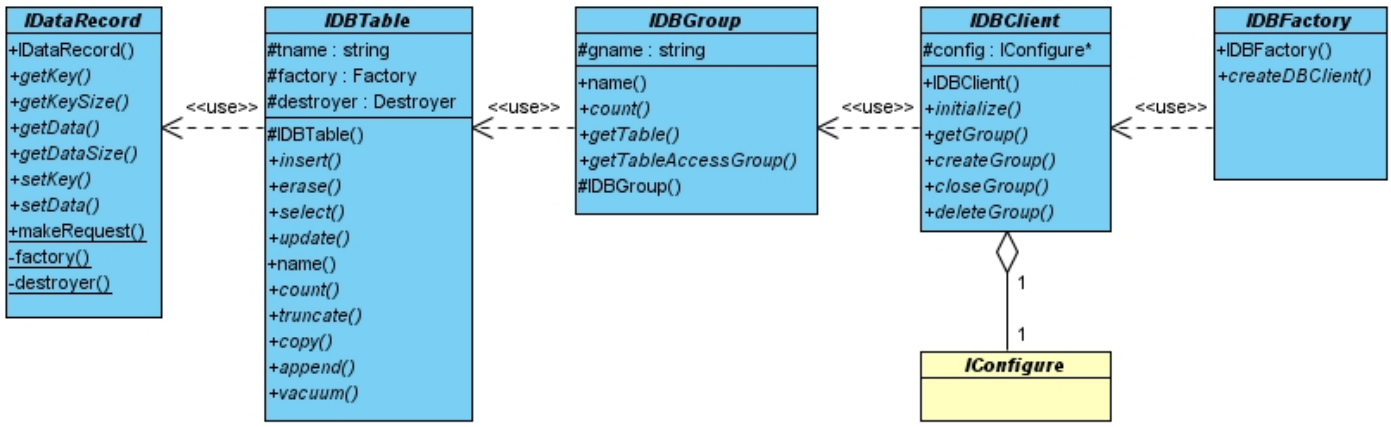


Figura 15: *Diagrama de Clases del paquete DBClient.*

Descripción de las clases del paquete DBClient	
Nombre de la clase	**Descripción**
IDataRecord	Las clases concretas que implementen esta interfaz constituyen los tipos de datos que serán manejados por cada plugin en sus tablas de BD.
IDBTable	Esta clase representa el concepto de tabla de BD. Expone un conjunto de funcionalidades que permiten interactuar con los datos almacenados.
IDBGroup	Esta clase representa el concepto de grupo de BD. Funciona como un contenedor que agrupa un conjunto de instancias concretas de IDBTable. El objetivo principal de esta clase radica en tener agrupadas a las tablas de BD que pertenecen a un plugin determinado, es decir, cada plugin posee su propio grupo de BD sobre el cual opera.
IDBClient	Esta clase representa la interfaz a implementar por clases controladoras de BD. Expone un grupo de funcionalidades para la gestión de grupos de BD, es decir, instancias concretas de IDBGroup.
IDBFactory	En la descripción del patrón de diseño *Abstract Factory* esta clase representa a la fábrica abstracta de la cual heredan las fábricas concretas de cada producto.

Tabla 14: *Descripción de las clases del paquete DBClient.*

3.2.7 Subpaquete BerkeleyDB.

Este subpaquete perteneciente al paquete DBClient, conforma la familia de objetos concretos para el SGBD embebido BerkeleyDB. El conjunto de clases que posee constituye la implementación de las interfaces definidas en el paquete DBClient.

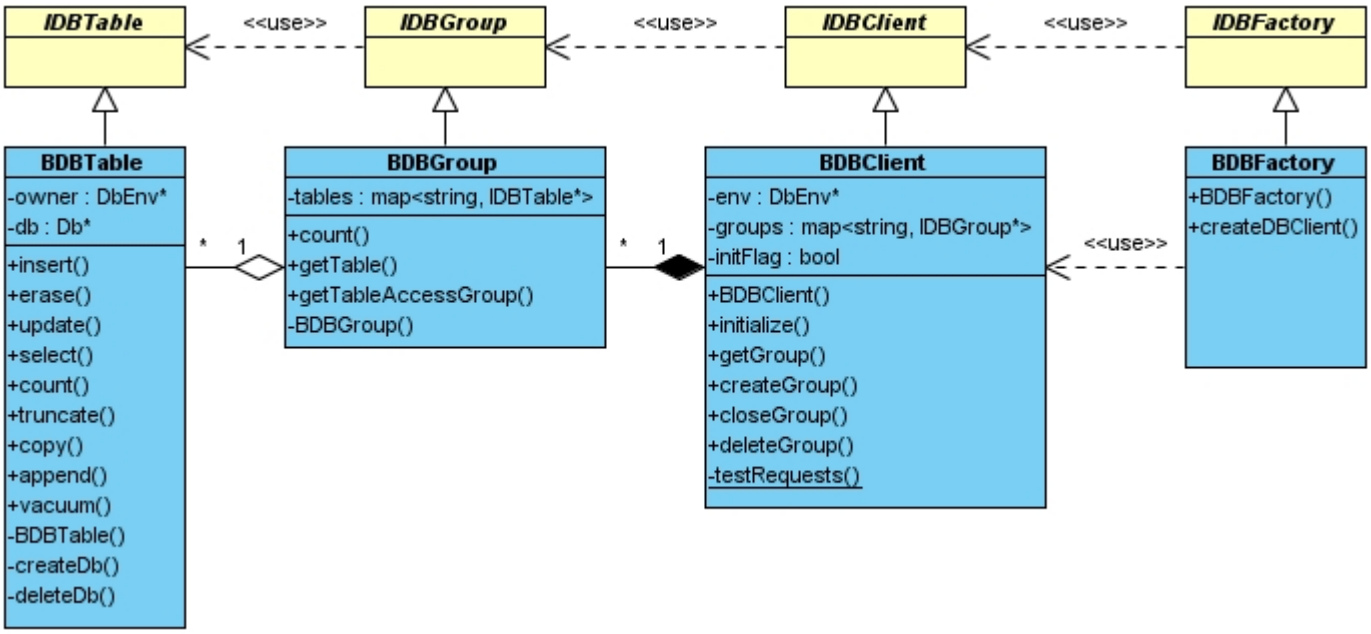


Figura 16: *Diagrama de Clases del subpaquete BerkeleyDB.*

3.3 Implementación.

3.3.1 Estilo de codificación.

Al programar es necesario seguir un determinado estilo que permita revisar, mantener y actualizar el código de una manera sencilla y ordenada. Seguir estas normas evita incurrir en errores y malas prácticas que dificultan la comprensión de las líneas de código. A continuación se muestra el conjunto de reglas seguidas para el estilo de código utilizado en la implementación del sistema.

Nombres.

- Los nombres de las clases son sustantivos singulares.

- Los nombres deben reflejar el qué y no el cómo.
- Los nombres no deben revelar detalles de implantación.
- Escoger nombres lo suficientemente largo para ser expresivos, pero evitando manejar nombres que dificulten la labor de implantación.
- Evitar nombres que permitan una interpretación subjetiva (evitar ambigüedad y asegurar abstracción).
- Evitar redundancia no repitiendo nombre de clases en sus elementos.
- Concatenar calificadores de cómputo a las variables que almacenen el producto de tal cómputo (avg, sum, min, max, index).
- Dado que los nombres generalmente son el producto de concatenar varias palabras, se debe emplear mayúscula para el inicio de cada palabra y minúscula para el resto, con excepción de la primera letra del nombre, la cual debe ser en minúscula.
- En el caso de las clases se utiliza la estructura anterior pero con excepción de que la primera letra del nombre debe ser en Mayúscula.
- Variables booleanas deben contener “is” en su nombre.
- Los nombres de constantes deben contener solo letras mayúsculas.
- Minimizar el uso de abreviaciones. En caso de ser requeridas, se debe ser consistente en su uso y cada abreviación debe significar solo una cosa. En general agregar a la documentación las abreviaturas.
- Los nombres de los métodos son frases que incluyen verbos.
- Los nombres de los atributos y parámetros son frases con sustantivos.
- Evitar el rehúso de nombres para distinto propósito.

Manejo de Errores.

- Se pueden manejar los errores mediante mecanismos de excepciones o mediante valores de retorno, aunque esto debe ser uniforme dentro de un mismo objeto.

- Es buena práctica emplear herramientas para identificar errores en la codificación en caliente.

Documentación y Comentarios.

- En el código debe documentarse en forma explicativa los pasos que se van ejecutando.
- Emplear oraciones completas al documentar código.
- Documentar mientras se programa.
- Documentar cualquiera cosa que no sea obvia en el código.
- Documentar eliminación de errores y cambios sobre el código.
- Al modificar el código se deben actualizar todos los comentarios y documentación asociada.
- Documentar cada rutina agregando: nombre del desarrollador, fecha, parámetros de entrada, valores de retorno, precondiciones, post-condiciones, dependencia con otros métodos o funciones y descripción general del algoritmo. Además, de realizarse cambios al código, debe indicarse el nombre de la persona que realizó el cambio, la fecha y la descripción del cambio, comenzando desde el o los cambios más recientes.
- Evitar agregar comentarios al final de líneas de código, salvo en el caso de declaraciones. En este caso tales comentarios deben estar alineados.
- Antes de la entrega de la aplicación, eliminar todos los comentarios superfluos y/o temporales con la finalidad de evitar confusiones en su mantenimiento.

Codificación.

- Se establece un tamaño de identación estándar de cuatro espacios, sin tabulaciones. Alinear secciones del código.
- Alinear verticalmente llaves de apertura y cierre.
- Usar espacios antes y después de los operadores que el lenguaje de programación permita.
- Emplear líneas en blanco para organizar el código, permitiendo crear párrafos de código para una mejor lectura.

- Evitar colocar más de una sentencia por línea.
- Emplear constantes en sustitución de números o cadenas de caracteres literales.
- Minimizar el alcance de las variables para evitar confusión y facilitar el mantenimiento.
- Emplear cada variable y rutina solo para un propósito.
- Evitar el uso de variables públicas, sustituirlas por variables privadas y métodos que provean el valor de tal variable, para mantener el encapsulamiento.
- Minimizar el uso de conversiones de tipo forzadas (castings), cuando se requiera su uso, debe ser comentada la justificación.
- Emplear select-case o switch en sustitución de if anidados sobre la misma variable.
- Liberar apuntadores de manera explícita.
- Emplear i, j, k, l, p, q, r para contadores en ciclos.
- Comentar siempre las llaves que cierran.
- Emplear al máximo operadores del tipo: +=, *=, /=, -=, ++, --, etc.
- Mantener la modularidad del código bajo el criterio de la lógica que encierra, no exagerar la modularidad.
- Emplear correctamente los tipos de ciclos: si es al menos una vez usar do-while, si es ninguna o más veces usar while-do, y si se conoce el número exacto de ciclos usar for.
- Inicializar todas las variables.
- Emplear líneas en blanco para separar pasos lógicos (declaraciones, lazos, etc.).
- Siempre asignar NULL a los apuntadores luego de ser destruidos (solo aplica para C).
- Evitar prácticas que incrementan explosivamente la complejidad, como lo son: objetos y variables globales y saltos tipo goto.

3.3.2 Diagrama de componentes.

Los diagramas de componentes muestran la organización y las dependencias entre el conjunto de componentes que integran el sistema desarrollado, así como la relación entre estos. Los componentes físicos pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente, entre otros. A continuación se muestra de forma general el diagrama de componentes del sistema, donde se expone la relación entre los diferentes tipos de componentes que lo conforman.

Los diagramas correspondientes a la distribución de las clases en ficheros de código fuente **.h** y **.cpp** pueden encontrarse en el Anexo 2 del documento.

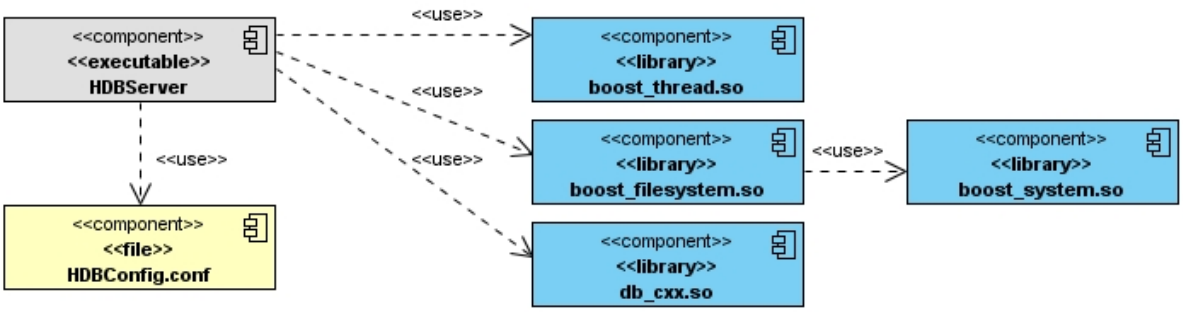


Figura 17: *Diagrama de componentes del sistema.*

El componente “HDBServer” con estereotipo <<*executable*>> representa el código fuente compilado en un ejecutable o binario que materializa las funcionalidades y responsabilidades anteriormente expuestas. El componente “HDBConfig.conf” de estereotipo <<*file*>> representa al fichero de configuración que carga el sistema al iniciarse, este archivo contiene datos de relevancia que permiten el correcto funcionamiento del sistema. Los componentes “boost_thread.so”, “boost_filesystem.so”, “boost_system.so” y “db_cxx.so” representan las bibliotecas necesarias para la ejecución del sistema.

3.4 Desarrollo de Plugins.

El desarrollo de plugins para el módulo de Gestión y Archivo de Datos se basa en la implementación y uso de un grupo de interfaces que fueron brevemente descritas en el epígrafe 3.2. Estas junto a las nuevas clases o subsistemas que se agreguen con las características particulares de diseño que posea cada

plugin, se distribuyen en código fuente que es compilado para la construcción de una biblioteca dinámica, la cual el sistema es capaz de usar en tiempo de ejecución, extendiendo así sus funcionalidades.

En el epígrafe 2.6 se expusieron de forma general algunas de las responsabilidades o enfoque de las funcionalidades que deberían poseer los plugins desarrollados, como la recepción, suministro y procesamiento de los datos históricos. El desarrollo e incorporación de subsistemas en los plugins, que realicen estas tareas constituyen aspectos particulares de cada desarrollador, debido a que cada uno se enmarca en los requisitos que debe cumplir. Por tal motivo solo se explicará el modo de usar las interfaces necesarias para la construcción de los plugins.

Un elemento esencial en la construcción de cada componente es la definición de los tipos de datos que va a manejar, ya sean variables, alarmas, eventos, entre otras. Estos tipos de datos constituyen los registros que se almacenan en la BD del plugin. Con este objetivo se utiliza la interfaz IDataRecord descrita anteriormente. Cada nuevo tipo creado que implemente esta interfaz podrá ser usado como registro en la BD. A continuación se describirán las funcionalidades de la interfaz que son necesarias especializar.

virtual void getKey(**void*** key) **const** = 0
Este método tiene como objetivo devolver en un arreglo de bytes el estado de los atributos del objeto que constituyan la llave del registro de BD. Como puede apreciarse la tarea de decidir que atributos forman parte de la llave del objeto visto como registro de BD y el proceso de serializarlos en el arreglo de bytes es tarea de cada desarrollador.
virtual unsigned int getKeySize() **const** = 0
Este método retorna la longitud en bytes que tendrá el arreglo de atributos llaves serializados.
virtual void getData(**void*** data) **const** = 0
Lo mismo que el método “getKey” pero con los atributos que constituyen la data del registro. Es importante destacar que un mismo atributo puede formar parte de la llave y de la data del registro a la vez.
virtual unsigned int getDataSize() **const** = 0
Este método retorna la longitud en bytes que tendrá el arreglo de atributos data serializados.

virtual int setKey(**const void*** key) = 0
A partir de un arreglo de bytes pasado por parámetros, este método debe ser capaz de restaurar el estado de los atributos llave de la clase.
virtual int setData(**const void*** data) = 0
A partir de un arreglo de bytes pasado por parámetros, este método debe ser capaz de restaurar el estado de los atributos data de la clase.

Tabla 15: *Descripción de los métodos abstractos de la interfaz IDataRecord.*

La segunda interfaz que se necesita implementar es “IPlugin”, la clase concreta que la implemente será la que exportará el plugin para que el sistema pueda interactuar con él. Es aconsejable por motivos de claridad que la clase concreta que implemente la interfaz tenga el nombre del plugin a desarrollar. A continuación se describirán en detalle los métodos de la interfaz “IPlugin” que son necesarios especializar.

virtual std::string name() **const** = 0
Este método retorna el nombre del plugin.
virtual const VendorInfo* getVendorInformation() **const** = 0
Este método retorna la dirección de la estructura “VendorInfo” que posee “IPlugin”. Esta estructura ofrece las características que posee el plugin como: su versión, el nombre del proveedor, y la descripción. Es responsabilidad del desarrollador dar los valores adecuados a la estructura.
virtual int getDBRequests(std::vector<IDataRecord::TableRequest>& requests) **const** = 0
Este método devuelve en el vector “requests” las solicitudes de tablas de BD que necesita el plugin para la gestión de sus datos, retornando un entero con la cantidad de solicitudes realizadas. Más adelante se explicará en detalle cómo realizar estas solicitudes.
virtual bool initialize(IDBGroup* group, **const** std::string configPath, **const** std::string cachePath,

std::string& error) = 0
Este método es invocado desde el sistema con el objetivo de inicializar el plugin. Si las solicitudes realizadas a través del método “getDBRequests” son aceptadas por el sistema, este le entrega al plugin, a través del parámetro “group” el grupo con las tablas de BD solicitadas. El sistema también entrega al plugin a través del parámetro “configPath” el directorio donde este guarda sus archivos de configuración y a través del parámetro “cachePath” un directorio de utilización opcional en caso de que el plugin guarde información temporal durante su ejecución. En caso de un fallo en la inicialización, el método retorna falso y una cadena con la descripción del error ocurrido a través del argumento “error”.
virtual void startService() = 0
Este método inicia la ejecución de los servicios que brinda el plugin. Normalmente cada plugin en este método inicia la ejecución de subsistemas relacionados con la recepción, suministro y procesamiento de datos históricos.
virtual void stopService() = 0
Este método detiene la ejecución de los servicios que brinda el plugin. Normalmente, es invocado por el sistema antes de cerrarlo o eliminarlo, por eso es necesario que el plugin detenga todos los subsistemas necesarios y guarde todos los estados necesarios que contribuyan a mantener la integridad de los datos que maneja.
virtual unsigned int dbTablesCount() **const** = 0
Este método retorna la cantidad de tablas de BD usadas por el plugin.

Tabla 16: *Descripción de los métodos abstractos de la interfaz IPlugin.*

La explicación de cómo realizar solicitudes de tablas de BD a través del método “getDBRequests” se hará mediante el siguiente ejemplo. Supongamos que se tienen las siguientes clases concretas de la interfaz “IDataRecord” que definen nuevos tipos de datos para un plugin determinado, se han omitido las definiciones de las clases para mayor brevedad:

```
class Point : public IDataRecord
{
/***/
};
```

```
class Alarm : public IDataRecord
{
/***/
};
```

```
class Event : public IDataRecord
{
/***/
};
```

Supongamos además que implementamos la interfaz “IPlugin” a través de la clase concreta “MyPlugin”. Una forma en la que podrían realizarse solicitudes de tablas de BD a través del método “getDBRequests” es mediante la siguiente implementación:

```
int MyPlugin::getDBRequests(std::vector<IDataRecord::TableRequest>& requests) const
{
    requests.clear();

    requests.push_back( IDataRecord::makeRequest<Point>("Point0") );
    requests.push_back( IDataRecord::makeRequest<Point>("Point1") );
    requests.push_back( IDataRecord::makeRequest<Alarm>("Alarm0") );
    requests.push_back( IDataRecord::makeRequest<Alarm>("Alarm1") );
    requests.push_back( IDataRecord::makeRequest<Event>("Event0") );
    requests.push_back( IDataRecord::makeRequest<Event>("Event1") );

    return requests.size();
}
```

Como puede observarse, para realizar una solicitud de tabla de BD se invoca la función genérica y estática “makeRequest” de la clase “IDataRecord”. En esta implementación se han insertado en el vector seis solicitudes: las dos primeras solicitan tablas para almacenar tipos de datos “Point”, con los nombres “Point0” y “Point1”, así sucesivamente se insertan dos solicitudes de tablas que manejen tipos de datos “Alarm” y otras dos para el tipo de datos “Event”.

Como parte de la implementación del método “initialize” de la interfaz “IPlugin” se debe actualizar el atributo miembro privado “group” con el grupo de BD recibido como argumento. Para acceder a algunas de las tablas que tiene el grupo, solo es necesario invocar su función miembro “getTable” con el nombre de la tabla como parámetro, luego puede utilizarse en la aplicación de alguna de sus operaciones definidas. Apoyado en las definiciones del ejemplo anterior, el siguiente fragmento de código ilustra de forma muy sencilla lo explicado.

```
Point myPoint;

IDBTable *table = group->getTable("Point0");

table->insert(&myPoint, IDBTable::OVERWRITE);
```

El último paso es definir la función registradora del plugin, que constituye el punto de entrada a la biblioteca, esta retorna a través de sus argumentos el nombre del plugin desarrollado y la dirección de la fábrica de instancias del mismo. En el Anexo 3 del documento se muestra el fragmento de código de la función registradora y la función fábrica, correspondientes al ejemplo anterior. El código debe ubicarse en un archivo fuente **.cpp**.

Para una mayor organización en el desarrollo de los plugins, se propone una estructura general de paquetes expresada a través del siguiente diagrama de componentes:

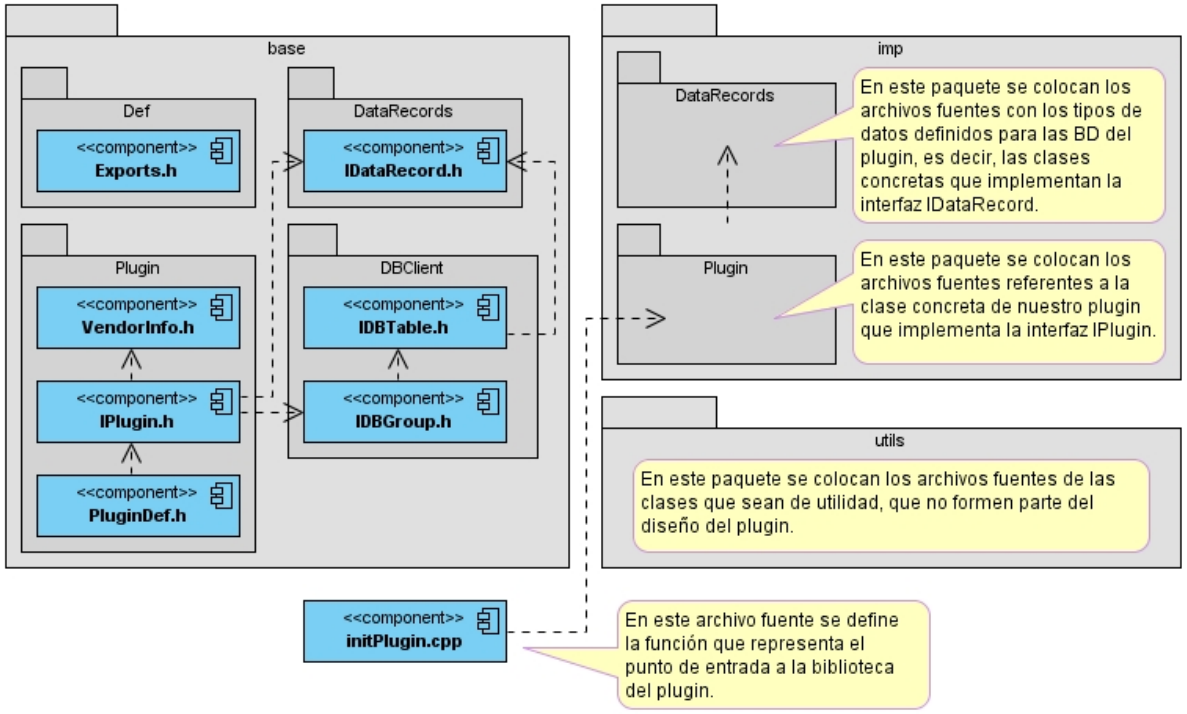


Figura 18: *Estructura de paquetes para el desarrollo de plugins.*

3.5 Pruebas.

La prueba del software constituye un elemento crítico para la garantía de la calidad. Los dos métodos de prueba más utilizados son el Método de Prueba de Caja Blanca y el Método de Prueba de Caja Negra. La diferencia entre ambos consiste en que con el método de Caja Blanca es necesario conocer el código y estar bastante relacionado con él para saber exactamente cuál es la lógica interna de lo que se va a

probar, sin embargo, en el método de Caja Negra solo basta con conocer las posibles entradas y salidas del programa. A continuación se realizan una serie de pruebas al sistema basadas en el método de Caja Negra.

Para la realización de pruebas al sistema se desarrollaron cuatro plugins: plTest1, plTest2, plTest3 y plScadaPoints, contenidos en las bibliotecas dinámicas plTest1.so, plTest2.so, plTest3.so y plScadaPoints.so respectivamente. Los tres primeros no poseen ningún subsistema para la recepción, suministro o procesamiento de datos, solo implementan las interfaces brindadas para el desarrollo de plugins y cada uno realiza tres solicitudes de BD. El cuarto plugin presenta un poco más de complejidad pues implementa internamente un subsistema para la recepción de puntos enviados por una aplicación de pruebas a través del Middleware, que es el módulo del SCADA encargado de la gestión de comunicación entre el resto de los módulos que lo integran, brindando un conjunto de interfaces que facilitan el intercambio de información.

Los primeros casos de prueba realizados fueron utilizando los plugins plTest1, plTest2, plTest3, estos fueron desarrollados con el objetivo de probar las funcionalidades generales del sistema, a través de la aplicación de comandos, los cuales pueden verse en el Anexo 4 del documento. A continuación se presentan los casos de pruebas realizados para los tres plugins mencionados.

Caso de prueba 1		
Acción del usuario.	**Respuesta esperada**	**Respuesta obtenida**
El operador coloca las bibliotecas plTest1.so y plTest2.so de los plugins plTest1 y plTest2 respectivamente en el directorio de plugins definido en el archivo de configuración. Luego se ejecuta el sistema por un terminal pasando como argumento el archivo de configuración.	El sistema deberá cargar las bibliotecas plTest1.so y plTest2.so, registrando los plugins plTest1 y plTest2, y poniendo en ejecución sus servicios.	El sistema cargó las bibliotecas plTest1.so y plTest2.so, registrando los plugins, plTest1 y plTest2, y poniendo en ejecución sus servicios. **Anexo 5**
Caso de prueba 2		
Acción del usuario.	**Respuesta esperada**	**Respuesta obtenida**

Con el sistema en ejecución el operador coloca la biblioteca plTest3.so del plugin plTest3 en el directorio de plugins y ejecuta el comando **load plTest3.so** con el objetivo de registrarlo en el sistema.	El sistema deberá cargar la biblioteca plTest3.so, registrando el plugin plTest3.	El sistema cargó la biblioteca plTest3.so, registrando el plugin plTest3 satisfactoriamente. **Anexo 6**
Caso de prueba 3		
Acción del usuario.	**Respuesta esperada**	**Respuesta obtenida**
Con el sistema en ejecución se ejecuta el comando **plugins** con el objetivo de listar los plugins registrados.	El sistema deberá listar todos los plugins que tiene registrados, en este caso plTest1, plTest2 y plTest3, mostrando sus nombres, el de las bibliotecas que los contienen, el número de BD que usa cada uno y el estado en que se encuentran.	El sistema listó todos los plugins que tiene registrados. **Anexo 7**
Caso de prueba 4		
Acción del usuario.	**Respuesta esperada**	**Respuesta obtenida**
Se elimina el plugin plTest2 registrado en el sistema con el comando **delete plTest2**.	El sistema deberá eliminar la biblioteca plTest2.so del directorio de plugins, borrar las BD y los directorios de configuración y de cache pertenecientes al plugin plTest2.	El sistema eliminó la biblioteca y todos los datos físicos asociados al plugin plTest2. **Anexo 8**
Caso de prueba 5		
Acción del usuario.	**Respuesta esperada**	**Respuesta obtenida**
Con el sistema en ejecución se activa el servicio del plugin plTest3 mediante el comando **start plTest3**.	El sistema deberá poner en ejecución el servicio del plugin plTest3.	El sistema ejecutó el servicio del plugin plTest3. **Anexo 9**
Caso de prueba 6		
Entrada	**Respuesta esperada**	**Respuesta obtenida**

Con el sistema en ejecución se detiene el servicio del plugin plTest1 mediante el comando **stop plTest1**.	El sistema deberá detener la ejecución de servicio del plugin plTest1.	El sistema detuvo el servicio del plugin plTest1. **Anexo 10**
Caso de prueba 7		
Entrada	**Respuesta esperada**	**Respuesta obtenida**
Con el sistema en ejecución se cierra el plugin plTest3 mediante el comando **close plTest3**.	Primeramente el sistema deberá detener la ejecución de servicio del plugin plTest3, luego cerrar sus tablas de BD y su registro.	El sistema detuvo el servicio de ejecución del plugin plTest3, cerró sus BD y su registro satisfactoriamente. **Anexo 11**
Caso de prueba 8		
Entrada	**Respuesta esperada**	**Respuesta obtenida**
Con el sistema en ejecución se consulta la versión del plugin plTest1 mediante el comando **about plTest1**.	El sistema deberá mostrar por la terminal la versión del plugin plTest1, el nombre de su desarrollador y la descripción.	El sistema muestra la versión, el nombre del desarrollador y la descripción del plugin. **Anexo 12_**
Caso de prueba 9		
Entrada	**Respuesta esperada**	**Respuesta obtenida**
Con el sistema en ejecución y el plugin plTest1 registrado, se aplica el comando **exit** con el objetivo de cerrar el sistema.	El sistema antes de cerrar deberá cerrar las BD y el registro del plugin plTest1.	El sistema cerró las BD y el registro del plugin plTest1 y luego cerró satisfactoriamente. **Anexo 13**

Tabla 17: *Primera parte de las pruebas realizadas al sistema.*

La segunda parte de las pruebas realizadas fueron utilizando el plugin plScadaPoints contenido en la biblioteca dinámica plScadaPoints.so, cuyo objetivo fue explicado anteriormente. La siguiente figura ilustra de manera general su funcionamiento.

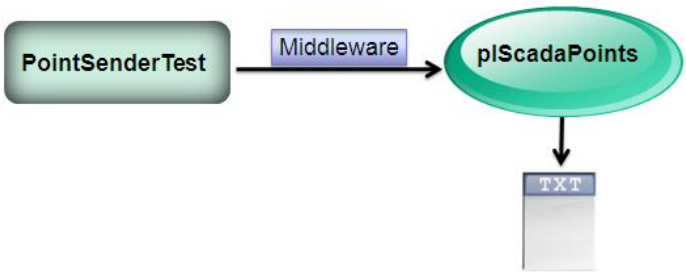


Figura 19: *Esquema general del funcionamiento de plugin de pruebas plScadaPoints.*

Como precondición para la ejecución de la siguiente prueba, el módulo de Middleware del SCADA debe estar en ejecución. Luego en el escenario se encuentra la aplicación de pruebas PointSenderTest. Esta al ejecutarse realiza el envío de un millón de puntos con frecuencia de un milisegundo, estos deben ser capturados por el plugin plScadaPoints, el cual al cerrarse, ya sea por la acción directa del operador o por la acción de cerrar el sistema, vuelca todo su contenido sobre un archivo de texto para verificar si la recepción de datos fue correcta, este archivo se localizará en el directorio cache del plugin.

Caso de prueba 10		
Acción del usuario.	**Respuesta esperada**	**Respuesta obtenida**
El operador coloca la biblioteca plScadaPoints.so que contiene el plugin plScadaPoints en el directorio de plugins definido en el archivo de configuración. Se ejecuta el sistema y luego se ejecuta aplicación de pruebas PointSenderTest. Para concluir se cierra la aplicación usando el comando **exit**.	En el directorio cache del plugin deberá encontrarse un archivo de texto generado por este, de nombre plScadaPoints.txt, con todos los puntos recibidos por el plugin, que deben coincidir con la cantidad de puntos enviados por la aplicación de pruebas PointSenderTest.	Se generó el archivo de texto plScadaPoints.txt con todos los puntos recibidos por el plugin, los cuales coinciden con los puntos enviados por la aplicación de pruebas PointSenderTest. **Anexo 14**

Tabla 18: *Segunda parte de las pruebas realizadas al sistema.*

3.6 Conclusiones del capítulo.

Con el desarrollo de este capítulo, se diseñaron las clases y paquetes que estructuran al Módulo de Gestión y Archivo de Datos. Fueron implementadas todas las clases de diseño, teniendo como resultado un producto que cumple con los requisitos funcionales descritos en el capítulo anterior. Se pudieron apreciar los elementos a tener en cuenta para el desarrollo de plugins, y como último paso se realizaron un conjunto de pruebas al sistema que validaron su correcto desempeño.

CONCLUSIONES

Una vez alcanzados los objetivos trazados para esta investigación, se puede arribar a las siguientes conclusiones:

1. Se desarrolló un módulo de Gestión y Archivo de Datos que permite la gestión de extensiones (plugins) que manejan datos históricos, permitiendo independizar cada solución desarrollada en un componente que se adapte a los requerimientos particulares de su escenario. La solución promueve la reutilización y el fácil mantenimiento.

2. Se obtuvo un producto fácil de desplegar y de operar, con mínimos recursos de memoria y procesamiento, pensado para escenarios que no involucren la manipulación de grandes volúmenes de datos y requieran de un rápido acceso a los mismos.

3. Ante un cambio de requisitos en el SCADA, que involucre modificaciones en su esquema de almacenamiento histórico, el módulo puede adaptarse rápidamente, con el desarrollo o ajuste de la extensión adecuada.

RECOMENDACIONES

Con vistas a mejorar la solución alcanzada se proponen las siguientes recomendaciones:

1. Desarrollar una o varias extensiones para la gestión de datos históricos del SCADA, como puntos, alarmas, eventos, entre otros.

2. Proveer al módulo de Gestión y Archivo de Datos de una interfaz visual, mediante el uso de algún *framework*, con el objetivo de hacer más amigable la interacción del operador con el sistema.

3. Desarrollar mecanismos que permitan incorporar al módulo los servicios de un SBDR.

REFERENCIAS BIBLIOGRÁFICAS

1. **Montero, Dagoberto, Barrantes, David B. y Quirós, Jorge M.** *Introducción a los sistemas de control supervisor y de adquisición de datos (SCADA).* Universidad de Costa Rica : s.n., 2004.

2. **Ferrari, Juan Pablo.** *Sistemas de Control Distribuido.* Universidad Nacional de Rosario : s.n., 2005.

3. **Jiménez Angulo, Verónica Consuelo.** *ESTUDIO DE FACTIBILIDAD DE UN SISTEMA DE COMUNICACIÓN INTEGRADO AL SISTEMA MICROONDA EXISTENTE, QUE SOPORTE APLICACIONES SCADA PARA EL POLIDUCTO ESMERALDAS – QUITO DE PETROCOMERCIAL.* Quito : s.n., 2008.

4. *SISTEMAS SCADA.* [En línea] 2 de marzo de 2006. [Citado el: 12 de abril de 2010.] http://www.automatas.org/redes/scadas.htm.

5. **De la Horra Diaz, Abdelaziz.** *Desarrollo del subsistema de comunicación para el Módulo Base de Datos de Históricos del proyecto SCADA Nacional.* Habana : s.n., 2008.

6. **Group, Object Management.** *Historical Data Access from Industrial Systems Specification.* 2005.

7. **Fundation, OPC.** *OPC Historical Data Access Specification.* 2003.

8. **Núñez Camallea, Noel L.** *GESTIÓN DE BASE DE DATOS CON ADO.NET.* Colombia : s.n., 2004.

9. **Alarcón Medina, José Manuel.** *Administración SGBD PostgreSQL.* 2006.

10. **Gil, Fidel, Albrigo, Javier y Do Rosario, Javier.** *SISTEMAS DE GESTIÓN DE BASE DE DATOS SGBD / DBMS.* Valencia : s.n., 2005.

11. *Servidores de bases de datos.* [En línea] Departamento Informática y Sistemas. Universidad de Murcia, 1 de septiembre de 2000. [Citado el: 7 de mayo de 2010.] http://www.um.es/docencia/barzana/DIVULGACION/INFORMATICA/sgbd.html.

12. **Murillo Alfaro, Félix.** *Herramientas Case.* 1999.

BIBLIOGRAFÍA CONSULTADA

1. *Librerías C/C++.* [En línea] ZATOR Systems, 9 de abril de 2010. [Citado el: 21 de mayo de 2010.] http://www.zator.com/libreriasC.htm.

2. *Los sistemas gestores de bases de datos y la red.* [En línea] La tecla de escape. [Citado el: 17 de abril de 2010.] http://latecladeescape.com/w0/basico/los-sistemas-gestores-de-bases-de-datos-y-la-red.html.

3. *Multi-Threaded Programming With POSIX Threads.* [En línea] Little Unix Programmers Group. [Citado el: 7 de mayo de 2010.] http://users.actcom.co.il/~choo/lupg/tutorials/multi-thread/multi-thread.html.

4. *Oracle Berkeley DB.* [En línea] Oracle. [Citado el: 23 de mayo de 2010.] http://www.oracle.com/database/berkeley-db/db/index.html.

5. *Oracle Berkeley DB 11g.* [En línea] Oracle. [Citado el: 23 de mayo de 2010.] http://www.oracle.com/technology/products/berkeley-db/index.html.

6. *Oracle Berkeley DB Products.* [En línea] Oracle. [Citado el: 7 de mayo de 2010.] http://www.oracle.com/database/berkeley-db/index.html.

7. *POSIX Threads Programming.* [En línea] LLNL. [Citado el: 7 de mayo de 2010.] https://computing.llnl.gov/tutorials/pthreads/.

8. *Sitio oficial de mcobject.* [En línea] [Citado el: 7 de mayo de 2010.] http://www.mcobject.com/.

9. *Sitio oficial de SQLite.* [En línea] [Citado el: 7 de mayo de 2010.] http://www.sqlite.org/.

10. *solidDB product family.* [En línea] IBM. [Citado el: 7 de mayo de 2010.] http://www-01.ibm.com/software/data/soliddb/.

11. **Camps Paré, Rafael, y otros.** *Base de Datos.* Catalunya : s.n., 2005.

12. **Chavarría Meza, Luis Eduardo.** *SCADA SYSTEM´S & TELEMETRY.* Mexico : s.n., 2007.

13. **Ezequiel Rozic, Sergio.** *BASE DE DATOS Y SU APLICACIÓN CON SQL.* Buenos Aires, Argentina : s.n., 2004.

14. **Gómez Ballester, Eva, y otros.** *Bases de Datos 1.* Universidad de Alicante : s.n., 2007.

15. **Group, Object Management.** *Data Acquisition from Industrial Systems Specification.* 2005.

16. **Mullins, Craig S.** *Empres Empress Offers an Effective Embedded Database Solution.* 2005.

17. **Olofson, Carl W.** *Embedded Database: The Invisible Engine That Could.* 2005.

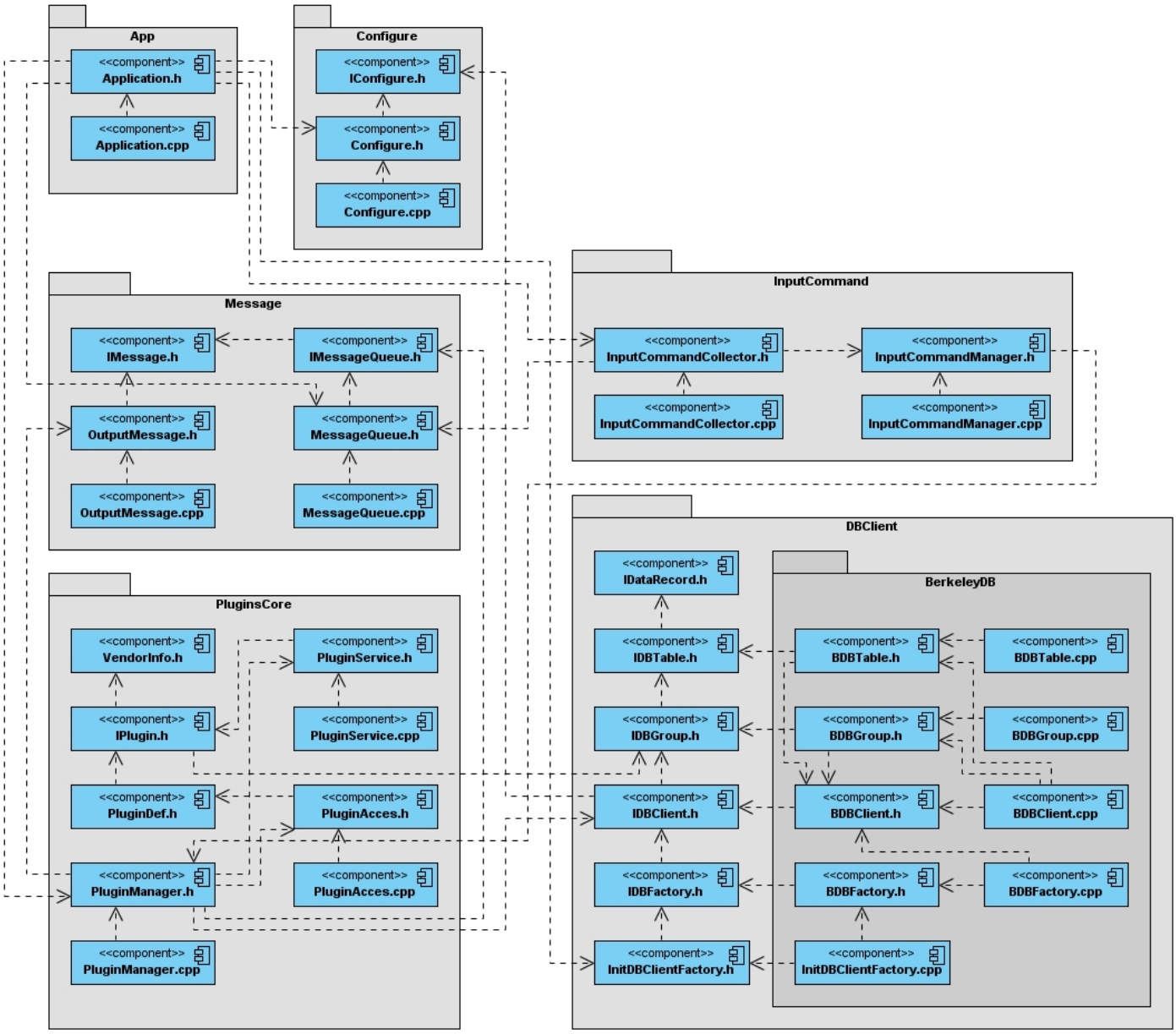
18. **Romagosa Cabús, Jaume, Gallego Navarrete, David y Pacheco Porras, Raúl.** *ETI, Especialidad en Electrónica Industrial.* Universidad Politécnica de Cataluña : s.n., 2004.

ANEXOS

Anexo 1: Agregados definidos en la especificación OPC HDA.

Comando	Descripción
INTERPOLATIVE	Se utiliza para recuperar los valores interpolados.
TIMEAVERAGE	Promedio de tiempo ponderado durante el intervalo de muestreo.
TOTAL	Valor total a lo largo del intervalo de muestreo.
AVERAGE	Promedio del valor de los datos.
COUNT	Número de valores disponibles en el intervalo de muestreo.
STDEV	Desviación estándar durante el intervalo de muestreo.
VARIANCE	Varianza durante el intervalo de muestreo.
MINIMUM ACTUAL TIME	Valor mínimo con esta estampa de tiempo.
MINIMUM	Valor mínimo.
MAXIMUM ACTUAL TIME	Valor máximo con esta estampa de tiempo.
MAXIMUM	Valor máximo.
START	Valor al inicio del intervalo de muestreo.
END	Valor al final del intervalo de muestreo.
DELTA	Diferencia entre el primer y el último valor.
REGSLOPE	Pendiente de la línea de regresión.
REGCONST	Intercepción de la línea de regresión al comienzo del intervalo de muestreo.
REGDEV	Desviación estándar de la línea de regresión.
RANGE	Diferencia entre los valores mínimo y máximo.
DURATION GOOD	Duración del tiempo en el intervalo durante el cual los datos son buenos.
DURATION BAD	Duración del tiempo en el intervalo durante el cual los datos son malos.
PERCENT GOOD	Porcentaje de datos en el intervalo que tienen buena calidad.
PERCENT BAD	Porcentaje de datos en el intervalo que tienen mala calidad.
WORST QUALITY	Peor calidad de los datos en el intervalo.

Anexo 2: Diagrama de componentes del sistema.



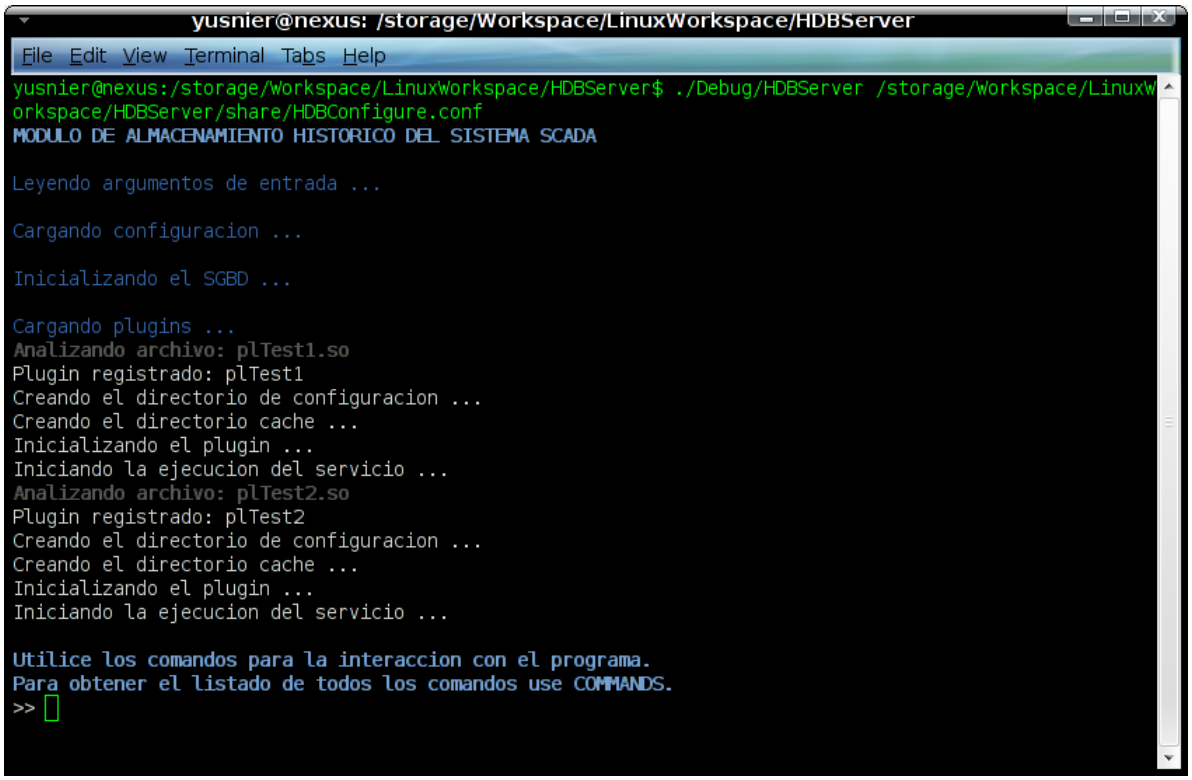
Anexo 3: Código fuente de la función registradora para el plugin “MyPlugin”.

```
#include <cstring>

#include "base/Def/Exports.h"
#include "base/Plugin/PluginDef.h"
#include "imp/Plugin/MyPlugin.h"

/* Se define la fábrica de para el plugin desarrollado */
IPlugin* createMyPlugin()
{
    return new MyPlugin();
}

/* Función registradora del plugin*/
extern "C" EXPORT_DIRECTIVE void exportPlugin(char* pluginName, PluginFactoryFunc* factory)
{
    std::strcpy(pluginName, "MyPlugin");
    (*factory) = createMyPlugin;
}
```

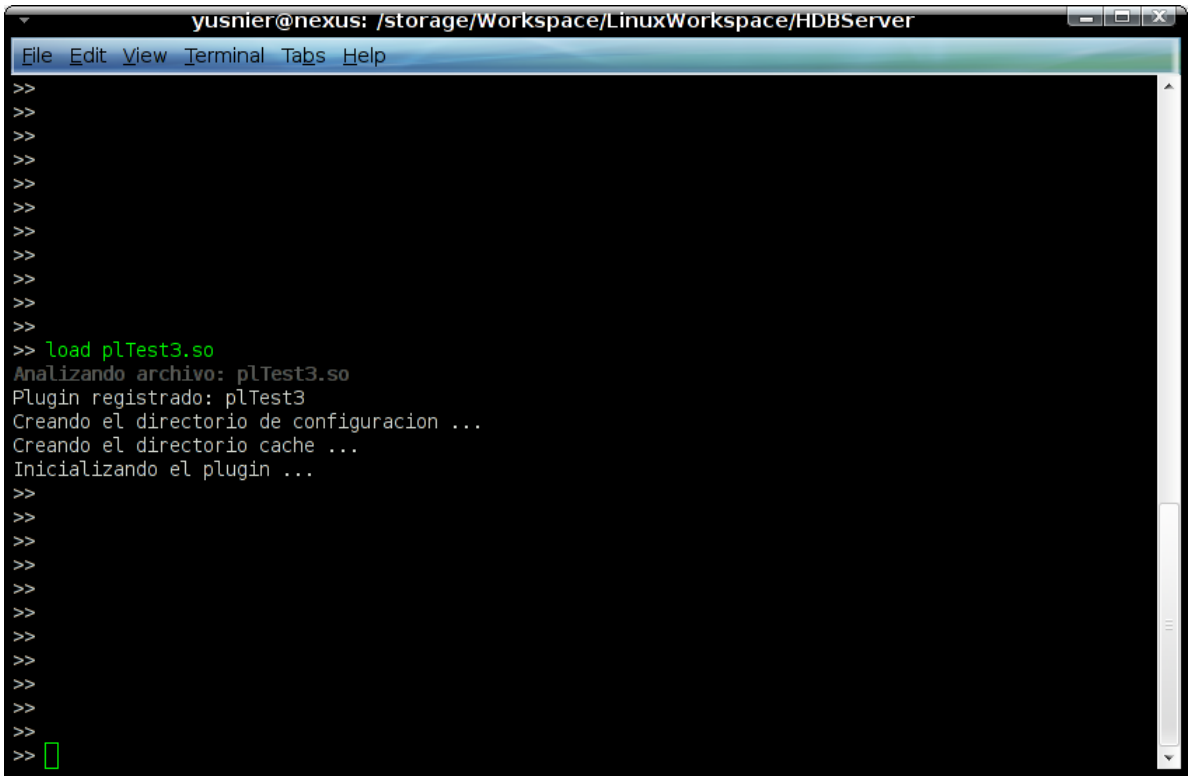
Anexo 4: Comandos aplicables al sistema desde el terminal o consola.

Comando	Descripción
COMMANDS	Lista todos los comandos con sus descripciones.
PLUGINS	Lista todos los plugins y sus estados.
EXIT	Finaliza la ejecución del programa.
LOAD <nombre de la biblioteca>	Registra el plugin de la biblioteca especificada.
START <nombre del plugin>	Activa la ejecución de servicio del plugin especificado.
STOP <nombre del plugin>	Detiene la ejecución de servicio del plugin especificado.
CLOSE <nombre del plugin>	Cierra el plugin especificado.
DELETE <nombre del plugin>	Elimina el plugin especificado y todos sus datos guardados.
ABOUT <nombre del plugin>	Muestra la información asociada al plugin especificado.

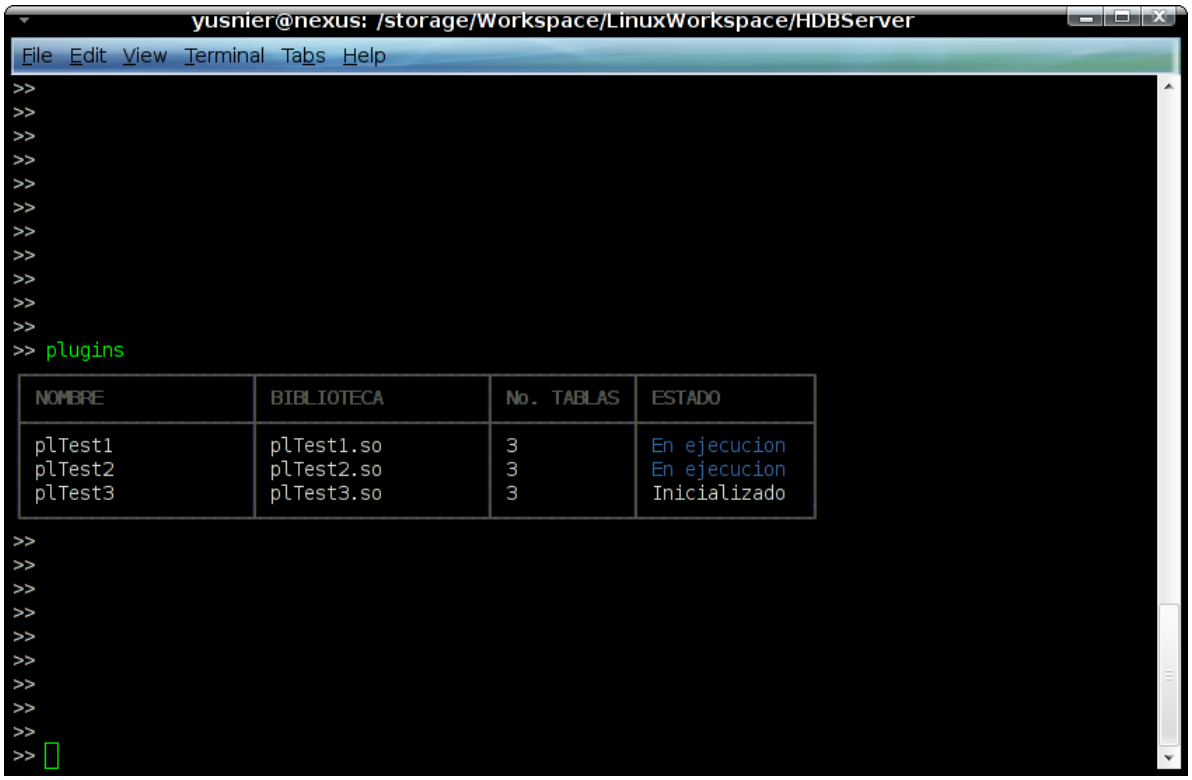
Anexo 5:


yusnier@nexus: /storage/Workspace/LinuxWorkspace/HDBServer
File Edit View Terminal Tabs Help
yusnier@nexus:/storage/Workspace/LinuxWorkspace/HDBServer$ ./Debug/HDBServer /storage/Workspace/LinuxWorkspace/HDBServer/share/HDBConfigure.conf
MODULO DE ALMACENAMIENTO HISTORICO DEL SISTEMA SCADA
Leyendo argumentos de entrada ...
Cargando configuracion ...
Inicializando el SGBD ...
Cargando plugins ...
Analizando archivo: plTest1.so
Plugin registrado: plTest1
Creando el directorio de configuracion ...
Creando el directorio cache ...
Inicializando el plugin ...
Iniciando la ejecucion del servicio ...
Analizando archivo: plTest2.so
Plugin registrado: plTest2
Creando el directorio de configuracion ...
Creando el directorio cache ...
Inicializando el plugin ...
Iniciando la ejecucion del servicio ...
Utilice los comandos para la interaccion con el programa.
Para obtener el listado de todos los comandos use COMMANDS.
>>

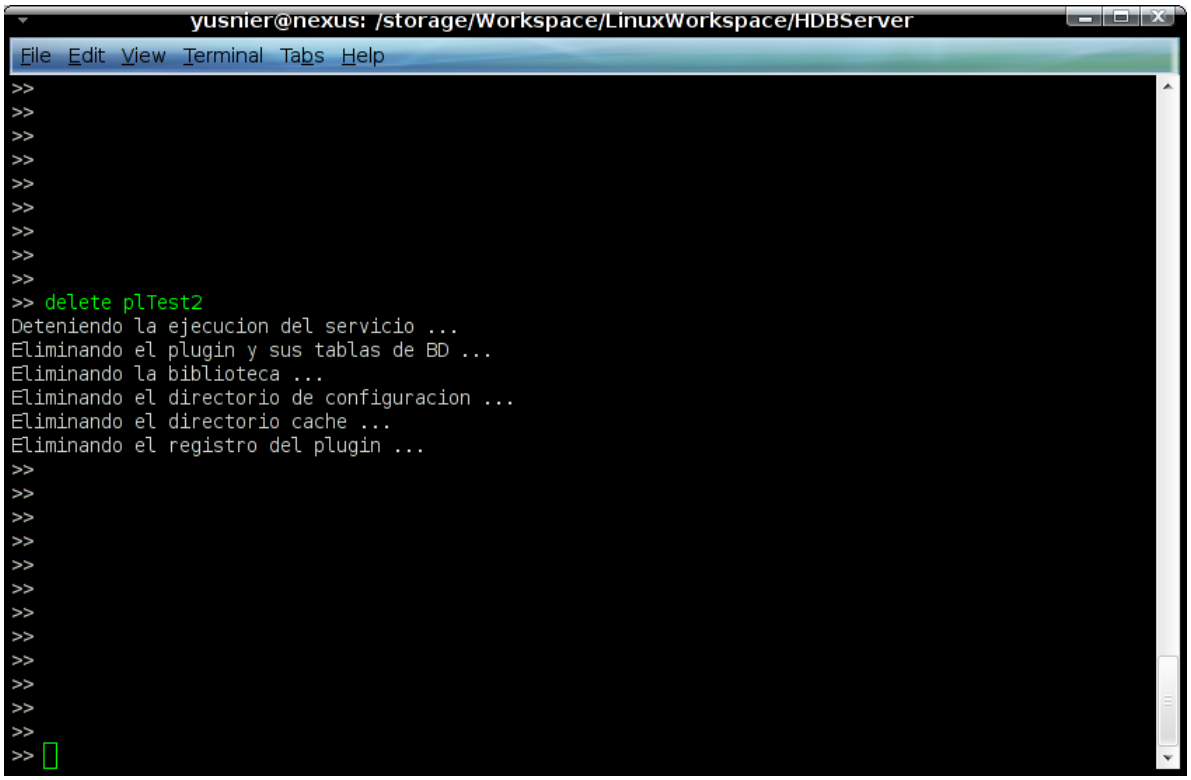
Anexo 6:


yusnier@nexus: /storage/Workspace/LinuxWorkspace/HDBServer
File Edit View Terminal Tabs Help
>> load plTest3.so
Analizando archivo: plTest3.so
Plugin registrado: plTest3
Creando el directorio de configuracion ...
Creando el directorio cache ...
Inicializando el plugin ...

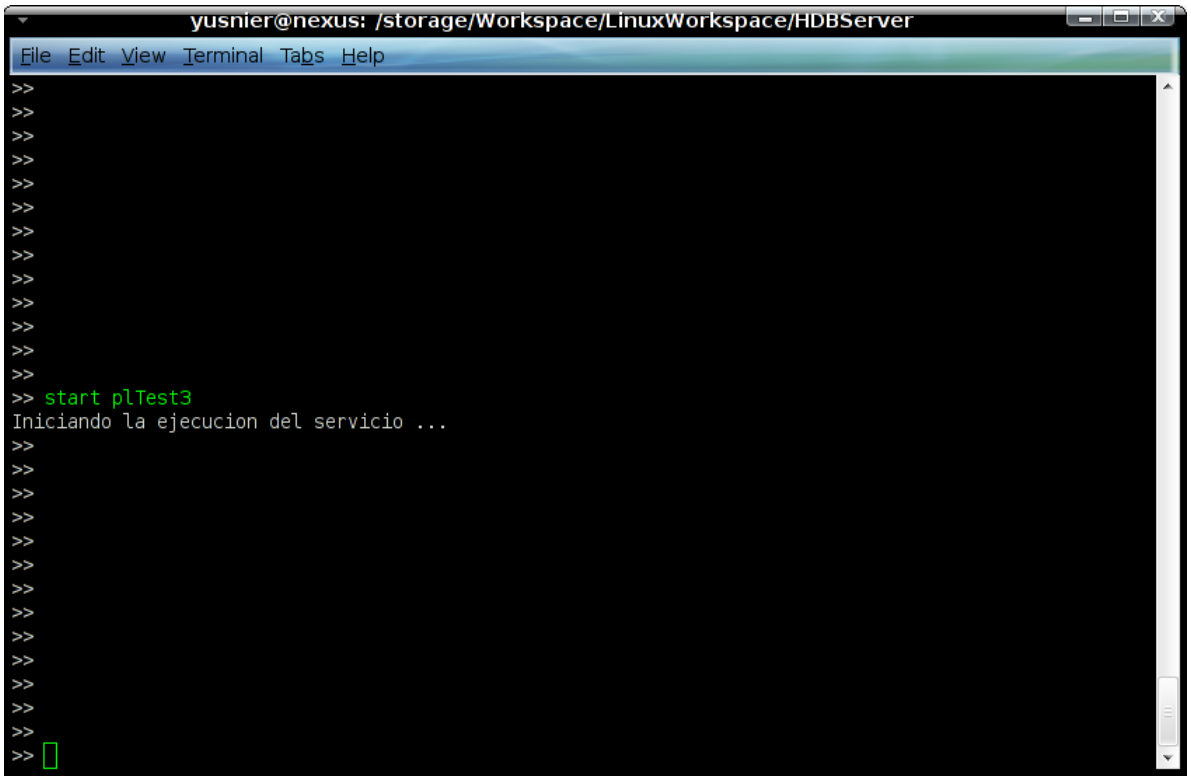
Anexo 7:


yusnier@nexus: /storage/Workspace/LinuxWorkspace/HDBServer
File Edit View Terminal Tabs Help
>> plugins
NOMBRE | BIBLIOTECA | No. TABLAS | ESTADO
plTest1 | plTest1.so | 3 | En ejecucion
plTest2 | plTest2.so | 3 | En ejecucion
plTest3 | plTest3.so | 3 | Inicializado

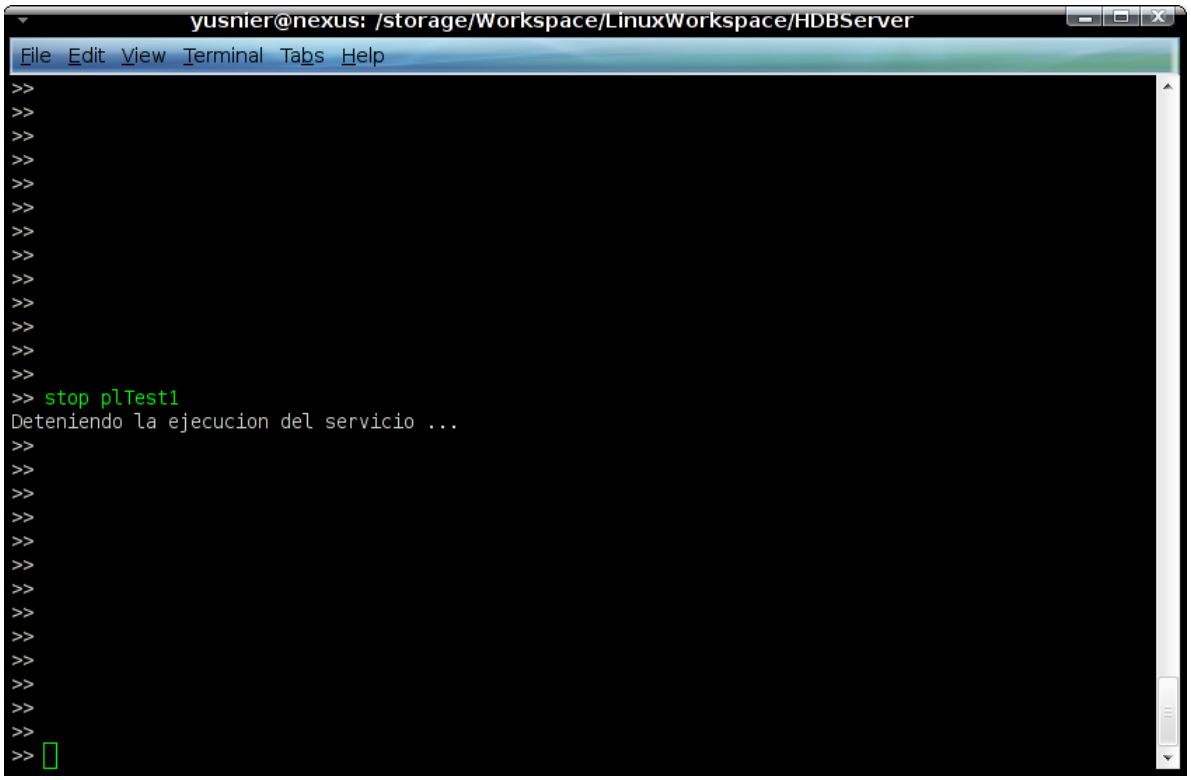
Anexo 8:


yusnier@nexus: /storage/Workspace/LinuxWorkspace/HDBServer
File Edit View Terminal Tabs Help
>> delete plTest2
Deteniendo la ejecucion del servicio ...
Eliminando el plugin y sus tablas de BD ...
Eliminando la biblioteca ...
Eliminando el directorio de configuracion ...
Eliminando el directorio cache ...
Eliminando el registro del plugin ...

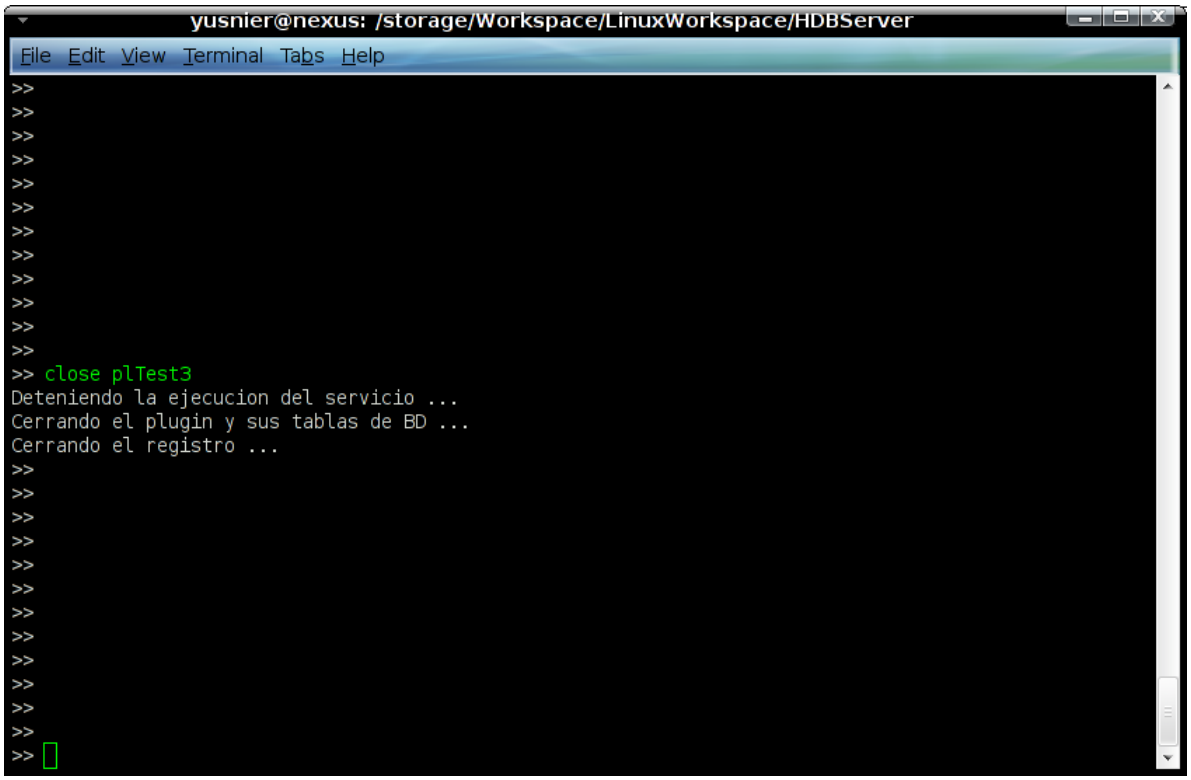
Anexo 9:


yusnier@nexus: /storage/Workspace/LinuxWorkspace/HDBServer
File Edit View Terminal Tabs Help
>> start plTest3
Iniciando la ejecucion del servicio ...

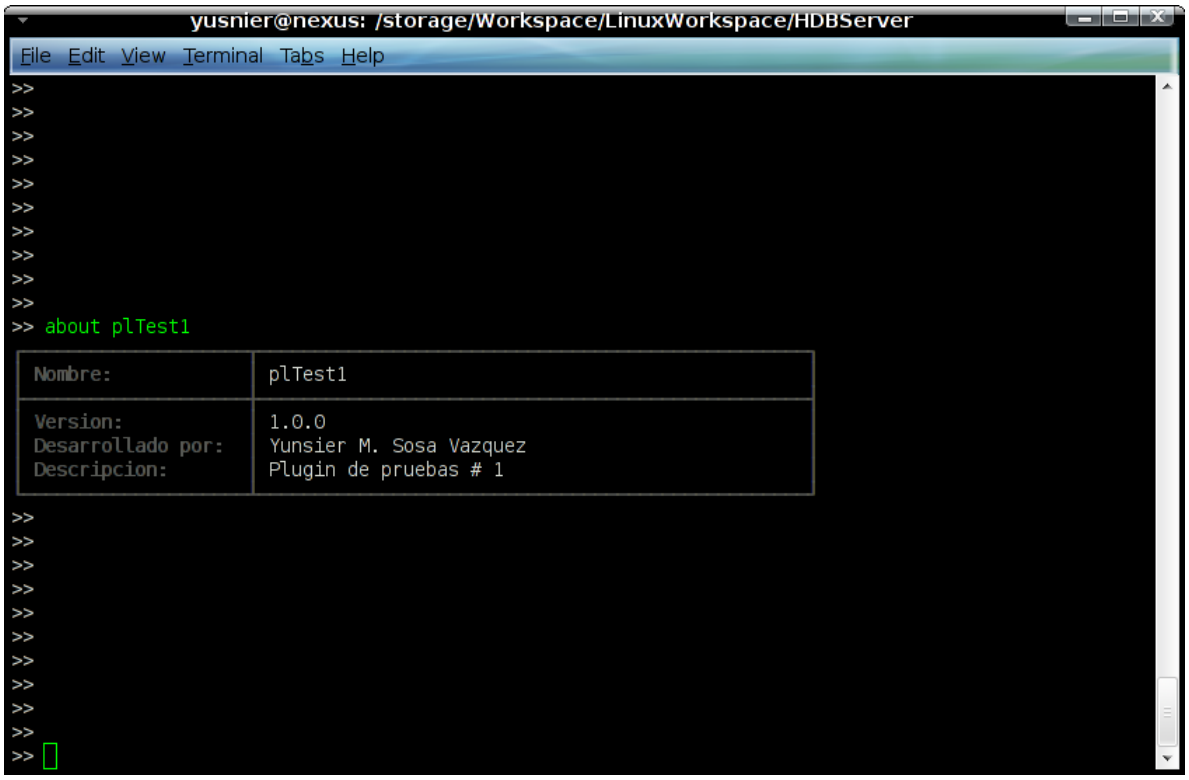
Anexo 10:


yusnier@nexus: /storage/Workspace/LinuxWorkspace/HDBServer
File Edit View Terminal Tabs Help
>> stop plTest1
Deteniendo la ejecucion del servicio ...

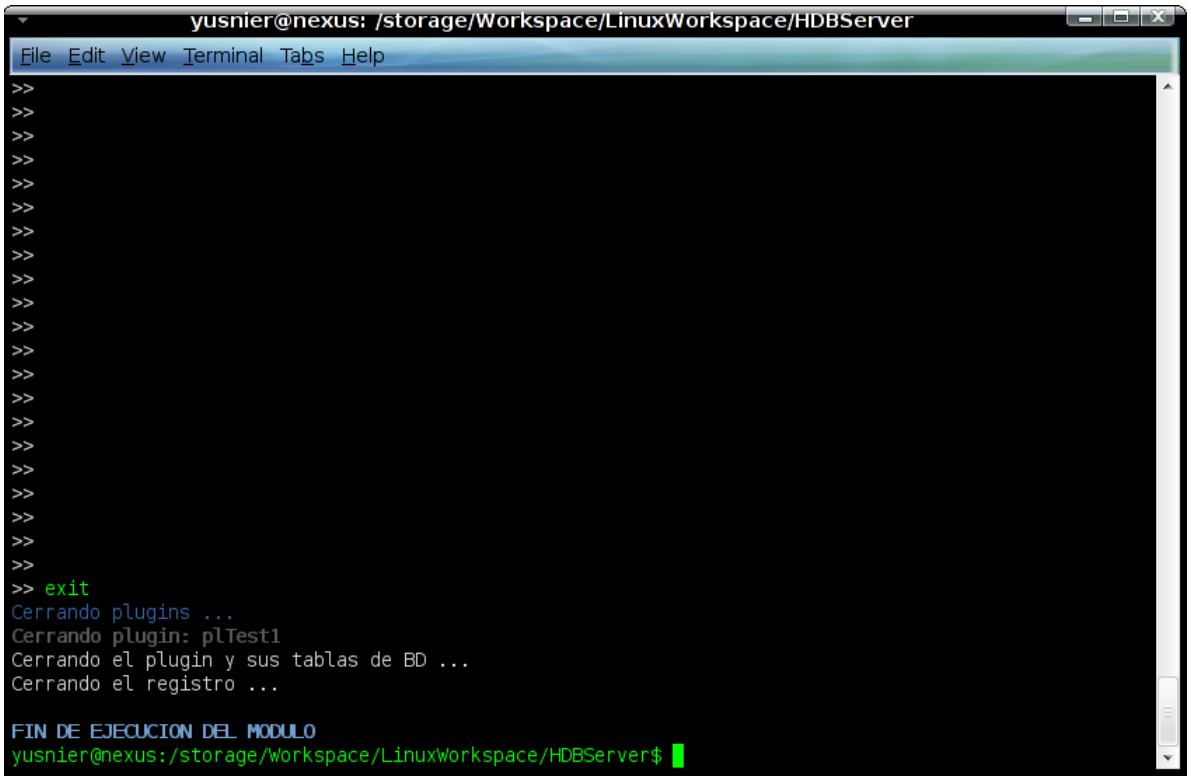
Anexo 11:


yusnier@nexus: /storage/Workspace/LinuxWorkspace/HDBServer
File Edit View Terminal Tabs Help
>> close plTest3
Deteniendo la ejecucion del servicio ...
Cerrando el plugin y sus tablas de BD ...
Cerrando el registro ...

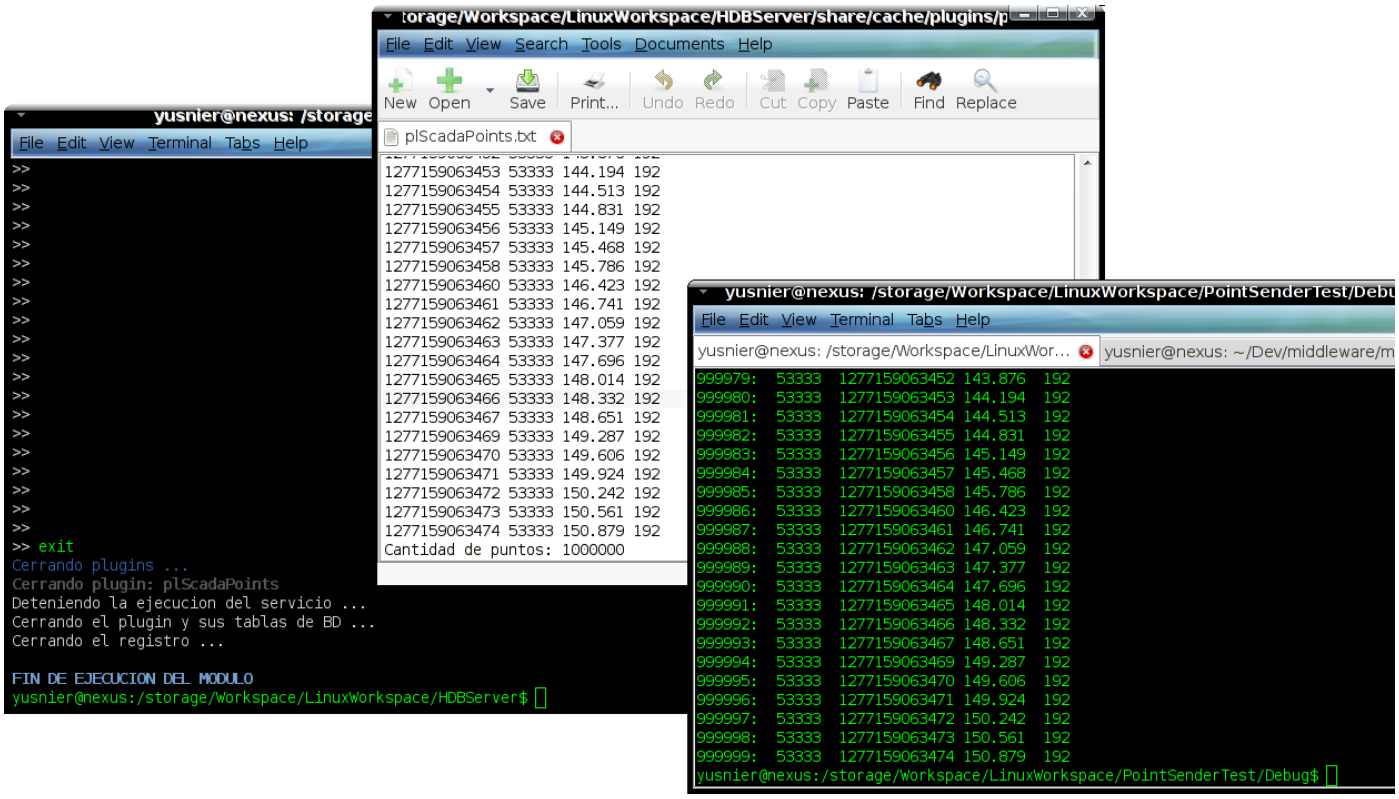
Anexo 12:


yusnier@nexus: /storage/Workspace/LinuxWorkspace/HDBServer
File Edit View Terminal Tabs Help
>> about plTest1
Nombre: plTest1
Version: 1.0.0
Desarrollado por: Yunsier M. Sosa Vazquez
Descripcion: Plugin de pruebas # 1

Anexo 13:


yusnier@nexus: /storage/Workspace/LinuxWorkspace/HDBServer
File Edit View Terminal Tabs Help
>> exit
Cerrando plugins ...
Cerrando plugin: plTest1
Cerrando el plugin y sus tablas de BD ...
Cerrando el registro ...
FIN DE EJECUCION DEL MODULO
yusnier@nexus:/storage/Workspace/LinuxWorkspace/HDBServer$

Anexo 14:


yusnier@nexus: /storage
File Edit View Terminal Tabs Help
>> exit
Cerrando plugins ...
Cerrando plugin: plScadaPoints
Deteniendo la ejecucion del servicio ...
Cerrando el plugin y sus tablas de BD ...
Cerrando el registro ...
FIN DE EJECUCION DEL MODULO
yusnier@nexus:/storage/Workspace/LinuxWorkspace/HDBServer$
torage/Workspace/LinuxWorkspace/HDBServer/share/cache/plugins/p
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
plScadaPoints.txt
1277159063453 53333 144.194 192
1277159063454 53333 144.513 192
1277159063455 53333 144.831 192
1277159063456 53333 145.149 192
1277159063457 53333 145.468 192
1277159063458 53333 145.786 192
1277159063460 53333 146.423 192
1277159063461 53333 146.741 192
1277159063462 53333 147.059 192
1277159063463 53333 147.377 192
1277159063464 53333 147.696 192
1277159063465 53333 148.014 192
1277159063466 53333 148.332 192
1277159063467 53333 148.651 192
1277159063469 53333 149.287 192
1277159063470 53333 149.606 192
1277159063471 53333 149.924 192
1277159063472 53333 150.242 192
1277159063473 53333 150.561 192
1277159063474 53333 150.879 192
Cantidad de puntos: 1000000
yusnier@nexus: /storage/Workspace/LinuxWorkspace/PointSenderTest/Debu
File Edit View Terminal Tabs Help
yusnier@nexus: /storage/Workspace/LinuxWor...
yusnier@nexus: ~/Dev/middleware/m
999979: 53333 1277159063452 143.876 192
999980: 53333 1277159063453 144.194 192
999981: 53333 1277159063454 144.513 192
999982: 53333 1277159063455 144.831 192
999983: 53333 1277159063456 145.149 192
999984: 53333 1277159063457 145.468 192
999985: 53333 1277159063458 145.786 192
999986: 53333 1277159063460 146.423 192
999987: 53333 1277159063461 146.741 192
999988: 53333 1277159063462 147.059 192
999989: 53333 1277159063463 147.377 192
999990: 53333 1277159063464 147.696 192
999991: 53333 1277159063465 148.014 192
999992: 53333 1277159063466 148.332 192
999993: 53333 1277159063467 148.651 192
999994: 53333 1277159063469 149.287 192
999995: 53333 1277159063470 149.606 192
999996: 53333 1277159063471 149.924 192
999997: 53333 1277159063472 150.242 192
999998: 53333 1277159063473 150.561 192
999999: 53333 1277159063474 150.879 192
yusnier@nexus:/storage/Workspace/LinuxWorkspace/PointSenderTest/Debug$

GLOSARIO DE TÉRMINOS Y ABREVIATURAS

Biblioteca: Es un conjunto de subprogramas para desarrollar software.

Buses: Sistemas digitales que transfieren datos entre los componentes de un ordenador o entre ordenadores.

CASE: (Del inglés, *Computer Aided Software Engineering*) Ingeniería de Software Asistida por Ordenador, son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software.

Comando: Instrucción u orden que el usuario proporciona a un sistema informático.

Framework: Es una estructura conceptual y tecnológica de soporte definida, normalmente, con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado.

LAN: (Del inglés, *Local Area Network*) red de área local, es la interconexión de varias computadoras y periféricos cuya extensión está limitada físicamente a un edificio o a un entorno generalmente no muy grande.

Módulo: Es una parte de un programa de ordenador. De las varias tareas que debe realizar un programa para cumplir con su función u objetivos, un módulo realizará una de dichas tareas (o quizá varias en algún caso).

PLC: (Del inglés, *Power Line Communications*) Controlador Lógico Programable, Es un equipo electrónico diseñado para programar y controlar procesos secuenciales en tiempo real.

Serialización: Es el proceso de convertir un objeto en una secuencia de bytes para conservarlo en memoria, una base de datos o un archivo. Su propósito principal es guardar el estado de un objeto para poder crearlo de nuevo cuando se necesita. El proceso inverso se denomina deserialización.

Sinóptico: Que presenta las partes principales de un asunto de manera clara, rápida y resumida.