Universidad de las Ciencias Informáticas
Facultad 5

Desarrollo del subsistema de comunicación para el Módulo Base de Datos de Históricos del proyecto "SCADA Nacional".

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Abdelaziz de la Horra Diaz.

Tutor: Ing. Fernando Jiménez López.

Co-tutor: Ing. Lazaro Abreu Reche.

Ciudad de la Habana

Julio 2008

"… el revolucionario verdadero está guiado por grandes sentimientos de amor…”

“Seamos realistas y hagamos lo imposible.”

Ernesto Che Guevara.

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 firmo la presente a los ____ días del mes de _____ del año _____.

Autor:	Tutor:
Abdelaziz de la Horra Diaz.	Ing. Fernando Jiménez López.
_____	_____

DATOS DE CONTACTO

Nombre y Apellidos: Fernando Jiménez López
Ciudadanía: cubano.
Institución: Universidad de las Ciencias Informáticas (UCI).
Título: Ingeniero en Informática.
Categoría Docente: Profesor Instructor.
E-mail: fjimenez@uci.cu

Graduado en Ingeniería Informática, en la Ciudad Universitaria José Antonio Echevarría (CUJAE). Actualmente Profesor Instructor de la Universidad de la Ciencias Informáticas impartiendo la asignatura de Gráficos por Computadora. Se desempeña como Jefe del Polo de Realidad Virtual.

AGRADECIMIENTOS

Mis más sinceros agradecimientos a todas aquellas personas que de una forma u otra contribuyeron a la realización de este trabajo. A la Revolución y a nuestro Comandante en Jefe Fidel, por haberme dado la oportunidad de ser un joven universitario de estos tiempos, superarme y convertirme un profesional al servicio de la Patria. Al Ing. Lazaro Abreu Reche, amigo que en todo momento me ofreció su apoyo y ayuda. A mi tutor, Ing. Fernando Jiménez López que siempre me guió y estuvo atento a darle respuesta a todas mis dudas durante el desarrollo de este trabajo. Un agradecimiento especial a la Lic. Leticia Rebeca León Diéguez por toda su ayuda.

A mi mis padres que me han dado todo lo que soy, por el cariño, la confianza y el apoyo que me han brindado durante toda mi vida. A Ali, mi niña, y a Danay por ser tan pacientes, por su cariño y su amor, razones que me dan fuerzas para seguir adelante aun estando siempre lejos de ellas. A toda mi gran familia que me ha apoyado en todos los momentos, buenos o malos. A todos mis amigos que siempre han estado ahí cuando los necesito.

DEDICATORIA

A mis padres, a mi niña, a Danay.
A toda mi familia.

RESUMEN

Desde hace años, las computadoras han ido ganando terreno en disímiles campos de la sociedad. Se destaca su uso en las tareas automatizadas de control y visualización de procesos industriales que anteriormente se efectuaban de forma manual y/o en Controladores Lógicos Programables. Con el auge de las tecnologías, se comenzaron a desarrollar complejos y útiles sistemas de control y adquisición de datos, en la cima de los cuales se ubica el SCADA.

Estos sistemas, por lo general, están formados por diferentes módulos que intercambian información entre ellos. Esto hace que la comunicación sea un factor determinante para su buen funcionamiento. Es por esto, que este trabajo aborda el diseño e implementación de un subsistema de comunicación, que le permita al Módulo Base de Datos de Históricos establecer un intercambio de datos con los demás módulos del proyecto “SCADA Nacional”.

Para cumplir con el objetivo del presente trabajo, se estudia la arquitectura propuesta para el sistema SCADA y los servicios que brinda el Módulo de Gestión y Archivo de Datos. Se analizan la interfaces que ofrece el *Middleware* para la transmisión de información, identificando además, las estructuras a transmitir. Se exponen las características técnicas que presentará el subsistema y se conciben los diagramas que permiten un mejor entendimiento del mismo. De este modo se sientan las bases para el diseño y la implementación de un subsistema de comunicaciones funcional.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La computadora personal se ha establecido, desde hace años, en un gran número de campos (oficina, hogar, industria, medicina, entre otros). Las tareas automatizadas de control y visualización que anteriormente se efectuaban de forma manual y/o en Controladores Lógicos Programables (*PLC*, por sus siglas en inglés), se comenzaron a realizar con sistemas automatizados basados en computadoras, utilizando tarjetas de expansión y/o de adquisición de datos, llegándose a desarrollar complejos y útiles sistemas. En la cima de estos sistemas se ubica el SCADA. [1]

Los sistemas SCADA han eliminado la necesidad de estar físicamente vigilando y ajustando los componentes de los procesos: una red de sensores transmite información del estado de los componentes a una sala de operadores que deciden si hay que realizar alguna modificación sobre el proceso. Muchas veces esta toma de decisiones está apoyada por una unidad central que descarga al operario de tareas repetitivas, dejándole actuar sobre el sistema a muy alto nivel.

En la Universidad de las Ciencias Informáticas (UCI), específicamente en la Facultad 5, se desarrolla el proyecto “SCADA Nacional” que tiene como objetivo el control de la producción de petróleo. Este sistema está conformado por diferentes módulos, cada uno de ellos con una función específica, que se comunican entre sí a través del módulo encargado del transporte de datos (*Middleware*).

Este intercambio de datos entre los módulos, constituye la columna vertebral del SCADA. En estos momentos el proyecto se ve altamente afectado por la inexistencia de comunicación del Módulo de Gestión y Archivo de Datos -conocido también como Base de Datos de Históricos (BDH)- con el resto de los módulos. Esta situación hace imposible el almacenamiento de los cambios de estado, lo cual conduce a un mal funcionamiento del sistema. De aquí surge, como **Problema Científico** a resolver: ¿Cómo establecer la comunicación entre el Módulo Base de Datos de Históricos y el resto de los módulos del proyecto “SCADA Nacional”, de modo que se garantice la transferencia eficiente de información entre ellos?

Para darle respuesta a esta interrogante, este trabajo tiene como **Objeto de Investigación** los subsistemas de comunicación de módulos de Gestión y Archivo de Datos de sistemas SCADA, centrándose específicamente en el subsistema de comunicación del Módulo Base de Datos de Históricos del proyecto “SCADA Nacional” como **Campo de Acción**.

El **Objetivo General** de la presente investigación es: desarrollar un subsistema del Módulo BDH capaz de establecer la comunicación eficaz entre éste y el resto de los módulos del proyecto “SCADA Nacional”.

A continuación se muestran las principales **Tareas de Investigación**:

- Estudio para la comprensión de la arquitectura propuesta para la aplicación SCADA, haciendo énfasis en el módulo BDH y su acoplamiento al sistema.
- Análisis de las interfaces que brinda el módulo *Middleware* para la transmisión de datos, para lograr una mayor comprensión de las mismas.
- Identificación de los formatos en que serán transmitidos los datos entre los módulos.
- Diseño de un subsistema o paquete encargado de la recepción y almacenamiento temporal de los datos.
- Diseño de un subsistema o paquete que tendrá la función de proveer el acceso a los datos almacenados en la Base de Datos (BD).
- Diseño de un subsistema o paquete cliente que será utilizado por los módulos que requieran consultar la BD.
- Implementación de los subsistemas de recepción y acceso a datos, así como el cliente.
- Integración de los subsistemas al módulo BDH del SCADA.

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 hace un bosquejo de la historia de los sistemas SCADA. Seguidamente se describen, enunciando sus principales características y funcionalidades, para luego hacer énfasis en las especificaciones del Módulo de Gestión y Archivo de Datos del proyecto “SCADA Nacional”, exponiendo los servicios que este debe brindar. En el capítulo 2: “Características del Sistema”, se ofrece una visión práctica del subsistema, exponiéndose las reglas del negocio, así como los requisitos funcionales y no funcionales, para así definir exactamente qué esperan los usuarios del subsistema. Finalmente se determinan los casos de uso y se describen los procesos de las principales funcionalidades. 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 subsistema de comunicaciones funcional.

Finalmente, como parte de los anexos se muestra un Glosario de Términos y Abreviaturas (*GTA*) que facilita la comprensión del lenguaje técnico y las abreviaturas utilizadas en el trabajo. Este Glosario muestra los términos en orden alfabético y pueden ser identificados en el cuerpo del documento por estar seguidos de las siglas (*GTA*).

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción.

El término SCADA proviene de las siglas de “Supervisory Control And Data Adquisition”. Traducido al español: Adquisición de Datos y Control Supervisor. Los sistemas SCADA utilizan la computadora y las 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 geográficamente dispersos, ya que pueden recolectar la información de una gran cantidad de fuentes rápidamente, y la presentan al 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. [2]

1.1 Historia.

Los primeros SCADA eran simplemente sistemas de telemetría (*GTA*), que proporcionaban reportes periódicos de las condiciones de campo vigilando las señales que representaban medidas y/o condiciones de estado en ubicaciones de campo remotas. Estos sistemas ofrecían capacidades muy simples de monitoreo y control, sin proveer funciones de aplicación alguna. La visión del operador en el proceso estaba basada en los contadores y las lámparas detrás de tableros llenos de indicadores. Mientras la tecnología se desarrollaba, las computadoras asumieron el papel de manejar la recolección de datos, disponiendo comandos de control, y una nueva función: presentación de la información sobre una pantalla de video. Las computadoras agregaron la capacidad de programar el sistema para realizar funciones más complejas de control. [2]

A partir de su surgimiento, los SCADA comenzaron a suplantar los sistemas de interfaz entre usuario y planta -basados en paneles de control repletos de indicadores luminosos, instrumentos de medición y botones pulsadores- por sistemas digitales que emulan estos mismos paneles sobre la pantalla del ordenador. Aunque el control directo lo continúan realizando mayormente los dispositivos de campo, estos se conectan ahora a la computadora, que realiza la función de presentación de la información de manera amigable, así como su tratamiento para la toma de decisiones, el control de la producción, etc. [1]

Los primeros sistemas automatizados SCADA fueron altamente modificados con programas específicos de aplicación para atender a requisitos de algún proyecto particular. Como consecuencia de que ingenieros de varias industrias asistieron al diseño de estos sistemas, su percepción de SCADA adquirió las características de su propia industria. Proveedores de sistemas de software SCADA, deseando reutilizar su trabajo previo sobre los nuevos proyectos, perpetuaron esta imagen de industria de acuerdo a su propia visión de los ambientes de control, con los cuales tenían experiencia. Solamente cuando nuevos proyectos requirieron funciones y aplicaciones adicionales, los desarrolladores de sistemas SCADA tuvieron la oportunidad de ganar experiencia en otras industrias.

Hoy, los proveedores de SCADA están diseñando sistemas que son pensados para resolver las necesidades de muchas industrias, con módulos de software específicos disponibles, para proporcionar las capacidades requeridas comúnmente. No es inusual encontrar softwares SCADA comercialmente disponibles, adaptados para procesamiento de papel y celulosa, industrias de aceite y gas, hidroeléctricas, gerenciamiento y provisión de agua, control de fluidos, etc. Puesto que los proveedores de SCADA aún tienen tendencia a favor de algunas industrias sobre otras, los compradores de estos sistemas a menudo dependen del proveedor para encontrar una solución comprensible a su requisito, y generalmente procuran seleccionar un vendedor que pueda ofrecer una completa solución con un producto estándar que esté apuntado hacia las necesidades específicas del usuario final. Si selecciona a un vendedor con experiencia limitada, el comprador debe estar preparado para asumir el esfuerzo de ingeniería necesario para poner el sistema en ejecución con éxito. [2]

1.2 Descripción de los sistemas SCADA.

SCADA es una aplicación de software y hardware especialmente diseñada para funcionar sobre ordenadores en el control de la producción, proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables (*GTA*), 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. [3]

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 y entre esta y el resto de las terminales remotas, a través de redes LAN (*GTA*) o WAN (*GTA*). 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.

1.2.1 Funcionalidades.

El paquete SCADA, como herramienta de interfaz humano-máquina, comprende una serie de funciones y utilidades encaminadas a establecer una comunicación lo más clara posible entre el proceso y el operador.

Entre las prestaciones de una herramienta de este tipo se destacan [4]:

- **Adquisición de datos:** Tarea que incluye la recolección, procesamiento primario y almacenamiento de la información.
- **Supervisión:** Monitoreo del funcionamiento del proceso, procesamiento estadístico de los datos y confección de reportes técnico-económicos. Además de ello, incluye alertas al operador sobre cambios detectados en la planta, tanto aquellos que no se consideren normales (alarmas) como cambios que se produzcan en su operación diaria (eventos). Estos cambios son almacenados en el sistema para su posterior análisis.
- **Control:** Actuación del operador sobre las variables del proceso para mantener su buen funcionamiento, tales como: abrir o cerrar válvulas, arrancar o parar bombas, etc.
- **Transmisión de datos:** Envío de la información de un nivel a otro del sistema o entre dispositivos ubicados a un mismo nivel.
- **Presentación de la información:** Viene dada por la interfaz humano-máquina y es la encargada de interactuar con los operadores, mantenedores, etc. En esta se incluye la muestra de reportes, gráficos de tendencias, historia de variables, entre otras.

1.2.2 Estructura.

Sobre la base de estas funcionalidades puede afirmarse que un SCADA posee dos grandes grupos de componentes: uno de software y otro de hardware.

Entre los de hardware pueden mencionarse [5]:

- Un ordenador central o Unidad Terminal Central (MTU,*"Master Terminal Unit"* en inglés)

- Ordenadores remotos (RTU, *"Remote Terminal Units"*, en inglés).
- Red de comunicación.
- Instrumentación de campo.

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 [3]:

- **Configuración:** permite al usuario definir el entorno de trabajo de su SCADA, adaptándolo a la aplicación particular que se desea desarrollar.
- **Interfaz gráfica del operador:** proporciona al operador las funciones de control y supervisión de la planta. El proceso se representa mediante gráficos sinópticos 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 preprogramadas 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 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. [6]

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 adaptivos. La utilidad más inmediata es la generación de reportes por parte de los operadores y usuarios del SCADA.

Existen varios tipos de históricos en la literatura de los sistemas SCADA, los más difundidos son los mencionados a continuación [6]:

- Servicio de históricos de variables (puntos).
- Servicio de históricos de alarmas.
- Servicio de históricos de eventos.
- Servicio de históricos de bitácora.

1.3.1 Servicios de históricos de variables.

Denominados por algunos autores Data Loggers, estos servicios se especializan en el manejo de los históricos de las variables del sistema, sean éstas 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:

- 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 (*GTA*) por intervalos de tiempo.

La figura 1 muestra un esquema que relaciona los diferentes módulos que intervienen en el proceso de un historiador de procesos. En ella se marcan 5 niveles, especializados en funcionalidades que van desde la captura de la información hasta la persistencia de los datos [6].

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 adquirir 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 entregar información histórica de la información 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 la recepción de la información del historiador, y almacenarla de manera segura, utilizando, opcionalmente, servicios de replicación (*GTA*). En este nivel se acostumbra a utilizar servidores de bases de datos que permitan servicios de réplica, respaldo, etc.

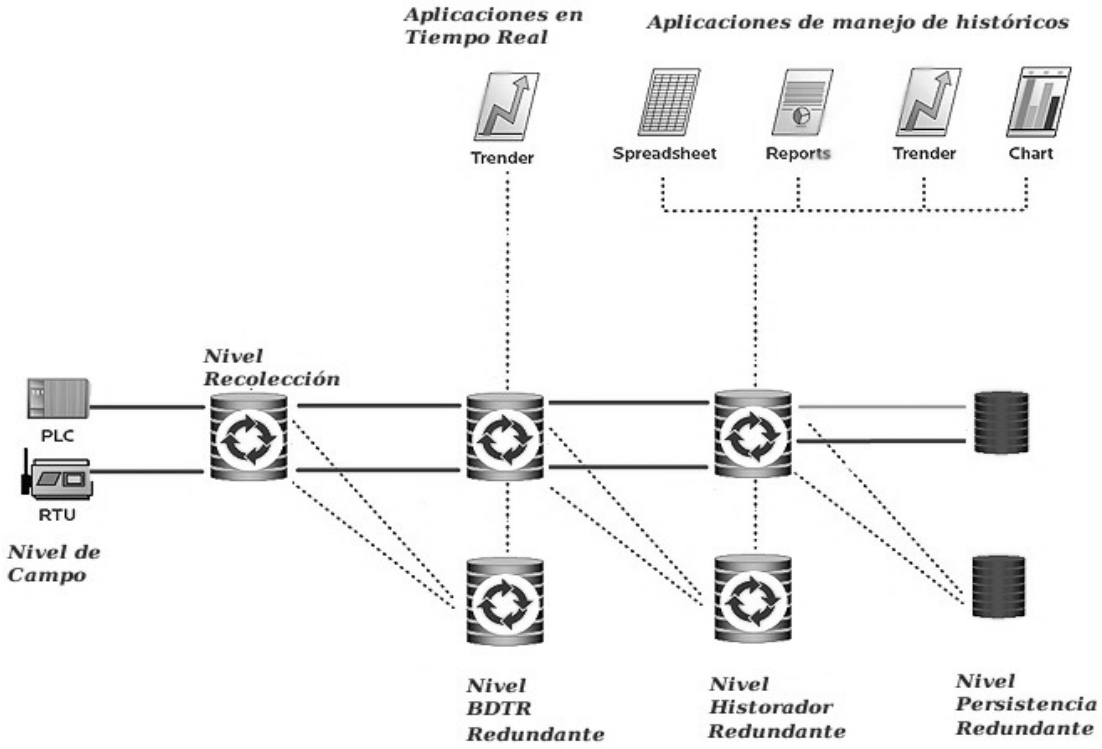

Figura 1: Esquema de comunicación entre los diferentes módulos que intervienen en el proceso de recolección y archivo de históricos.

Información que se envía a los históricos de variables.

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.

Al organizar la información para los históricos se requiere por cada variable la siguiente información. [6]

1. *VariableID*. Identificador de la variable en el Sistema.
2. *Value.* Estado (valor) de la variable en un instante de tiempo.
3. *Quality*. Calidad de la variable en un instante de tiempo (Buena, Mala o Incierta).
4. *TimeStamp.* Estampa de tiempo que marca el instante en que se capturó el dato.
5. *Alarmed.* Opcional, indicativo de si la variable se encuentra con alguna alarma activa. Es

común que se agregue como una sub-cualidad al punto.

En esquemas orientados a objetos, como los presentados en la especificación de históricos HDAIS [7], 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.

1.3.2 Servicios de históricos de alarmas.

Estos servicios se especializan 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. [6].

Por otro lado pueden haber 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 (como podría ser la caída de una red o al fallo catastrófico de recursos de hardware).

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

1. Históricos de ocurrencia de alarmas.
2. Históricos de seguimiento del manejo de alarmas.

Históricos de ocurrencia de alarmas. [8]

La ocurrencia de alarmas tiene una importancia trascendental para 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.

Información que se envía a los históricos de ocurrencia de alarmas.

En el SCADA se lleva un sumario de la ocurrencia y cambios en las alarmas, este sumario nutre la información, en tiempo real, de ocurrencia y manejo de las alarmas, requerida por las consolas de operación y por los históricos de alarmas.

Este sumario puede ser distribuido por todos los módulos generadores de alarmas o puede ser centralizado en un gestor de información que permita realizar búsquedas complejas relativas a las alarmas activas en un instante dado.

Dentro de los campos que requiere el sumario de alarmas se pueden mencionar los siguientes [6]:

1. *AlarmID*: Identificador de la alarma activada.
2. *AlarmZone*: Área a la cual pertenece la alarma (Nota: no es necesario transmitir este dato siempre, es definido por configuración y su relación con *AlarmID*).
3. *AlarmType*: Tipo de Alarma asociada (Nota: no es necesario transmitir este dato siempre, es definido por configuración y su relación con *AlarmID*).
4. *Description*: Descripción de la alarma asociada (Nota: no es necesario transmitir este dato siempre, es definido por configuración y su relación con *AlarmID*).
5. *Condition*: Condición que fue cumplida para activar la alarma (Nota: no es necesario transmitir este dato siempre, es definido por configuración y su relación con *AlarmID*).
6. *AsocResorceID*: Identificador del punto, dispositivo, módulo, etc. relacionado con la alarma activada
7. *AsocResorceType*: Identifica si es un punto, dispositivo, módulo, etc., lo que está relacionado con la alarma activada. Este campo se utiliza para realizar las búsquedas.
8. *Cause*: Define el valor del punto o trama de calidad que causó la alarma.
9. *Serverity*: Severidad de la alarma activada (Nota: no es necesario transmitir este dato siempre, es definido por configuración y su relación con *AlarmID*).
10. *ONTimeStamp*: Estampa de tiempo del instante en que se generó la alarma.
11. *OFFTimeStamp*: Estampa de tiempo del instante que cesó la alarma.
12. *ACKTimeStamp*: Estampa de tiempo del instante en que se reconoció la alarma.
13. *RESETTimeStamp*: Estampa de tiempo del instante que se hizo la acción de RESET a la alarma.
14. *State*: Si la alarma está activa, reconocida, etc.
15. *OcurrenceNumber*: Número de ocurrencias de la alarma.

Históricos de seguimiento del manejo de alarmas. [8]

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 estas se hayan detectado. Sin embargo, las tareas a ejecutar una vez detectada la alarma, por lo general son 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, paso sucesivo, debe tomar acciones para restaurar el estado normal del sistema, como por ejemplo, la modificación de algún parámetro de control.

Sin embargo, en ocasiones los operadores no cumplen con lo establecido. El primer incumplimiento es el no reconocimiento de la existencia de una alarma, indicativo de que el operador no realiza bien su trabajo. Por otra parte, existe la posibilidad de que el operador reconozca la alarma y no efectúe ninguna operación para eliminarla. Ambos comportamientos podrían desencadenar accidentes. La implementación de un servicio que garantice un seguimiento del manejo en las alarmas, permitirá prever futuros errores que pudieran llevar a pérdidas o ayudar en el peritaje de accidentes.

Se debe llevar un control estricto de las acciones sobre las alarmas, como reconocimiento, reinicio y volver al estado normal.

Información que se envía a los históricos en un seguimiento de alarmas.

El seguimiento de acciones sobre las alarmas, debe almacenar en los históricos la información que permita realizar auditorías de cada cambio que se realice sobre ellas. Las acciones de reconocer y reiniciar una alarma deben ser debidamente almacenadas de forma que puedan realizarse búsquedas avanzadas.

Comúnmente, estos eventos se colocan en los repositorios de datos que contienen no sólo la información del manejo de alarmas, sino de eventos que estén asociados a acciones auditables, como son la modificación de puntos, generación de reportes, etc. [6]

1. *AlarmID*: Identificador de la alarma que se modificó.

2. *AsocResourceID*: Identificador del recurso relacionado con la alarma.
3. *NewState*: Nuevo Estado al que se pasó la alarma, reconocida, activa, etc.
1. *Action*: Acción que produce el evento, reconocer, reiniciar, etc.
4. *UserID*, Identificador del usuario que modificó la variable.
5. *SourceID*: Identificador de la fuente de modificación, por ejemplo, Consola1.Despliege2
6. *TimeStamp*: Instante en que se produjo el cambio.

Esta información es considerada como “Eventos auditables del SCADA” (ver epígrafe: Servicios de históricos de eventos). Esto nos permite realizar una estructura de datos única para todos los eventos del sistema.

1.3.3 Servicios de históricos de eventos.

Estos 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 (*GTA*), fallas en la ejecución de un comando, caídas temporales de la red, etc. Estos permiten las auditorias de los diferentes servicios del SCADA. [6]

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.

Para mayor claridad se pueden dividir los eventos en las siguientes categorías [9]:

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

Cada una de estas categorías agrupa una serie de tipos de eventos. Cada uno de estos tipos debe ser diferenciado de sus similares, de todas las categorías, a través de un identificador único (*EventType*).

Un esquema de posible utilización seria identificar a cada tipo con un número único, donde cada

categoría estaría representada por un rango de identificadores, lo que permite realizar consultas de forma sencilla.

Información que se envía a los históricos de eventos.

Los eventos deben almacenar en los históricos los siguientes atributos [6] (6):

1. *EventDescription*: Cadena que contiene la información del evento ocurrido (por ejemplo: “Módulo de BDTR fuera de operación”).
2. *EventType*: Identificador único del tipo de evento.
3. *AsocResourceID*: Identificador del recurso asociado al evento (por ejemplo: Identificador del punto, dispositivo, etc).
4. *ResourceType*: Tipo de recurso asociado el evento. Esto se utiliza porque en muchos sistemas los identificadores de los recursos pueden ser repetidos en categorías diferentes, por ejemplo, un punto puede tener el mismo identificador de una alarma.
5. *UserID*: Identificador del usuario que produjo el evento.
6. *SourceID*: Identificador de la fuente del evento, por ejemplo: Consola1.Despliege2.
7. *TimeStamp*: Instante en que se produjo el evento.

1.3.4 Servicios de históricos de bitácoras.

Estos 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 podría ser interesante consultar por él o por los demás operadores en un futuro. [6]

Reutilizar las experiencias alcanzadas 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.

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.

Información que se envía a la bitácora.

Las bitácoras almacenan en los históricos los siguientes atributos [6]:

1. *Info*: Cadena que contiene la información que el usuario entiende debe ser almacenada en la bitácora, por ejemplo, “Al elevarse la temperatura del líquido a más de 30 grados, el controlador debe ser reajustado a los siguientes parámetros para obtener una respuesta adecuada”
2. *InfoType*: Tipo de información, se utiliza con fines de visualización y filtrado.
3. *UserID*: Identificador del usuario que produjo el registro en la bitácora.
4. *SourceID*: Identificador de la fuente del evento, por ejemplo: Consola1.Despliege2.
5. *TimeStamp*: Instante en que se produjo el evento.

1.3.5 Solicitud de información de los históricos.

Los usuarios, como consolas o interfaz de reportes, solicitarán al servicio de históricos que le entregue los datos, los cuales podrán ser filtrados por cualquiera de los campos existentes.

Solicitud de datos de las variables al histórico.

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, etc.

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 podemos mencionar los siguientes [6]:

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*. Así mismo, si no existe un valor exactamente en el *EndTime*, se ofrecerá el primer valor que se encuentre después este.

Aggregate: Este parámetro está inspirado en la especificación OPC HDA [10]. Básicamente define los métodos para realizar sumarios de la información del punto. Los 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”. La tabla 1 lista 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 de la consulta normalmente utilizado es SQL (*GTA*) o XQuery (*GTA*).

Tabla 1: Agregados definidos en la especificación OPC HDA.

Agregado	**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

1.4 Integración del módulo en la arquitectura del SCADA.

El gráfico que se muestra a continuación esquematiza el subsistema historiador, compuesto por un grupo de submódulos que permiten la interacción con los demás componentes del SCADA. [11]

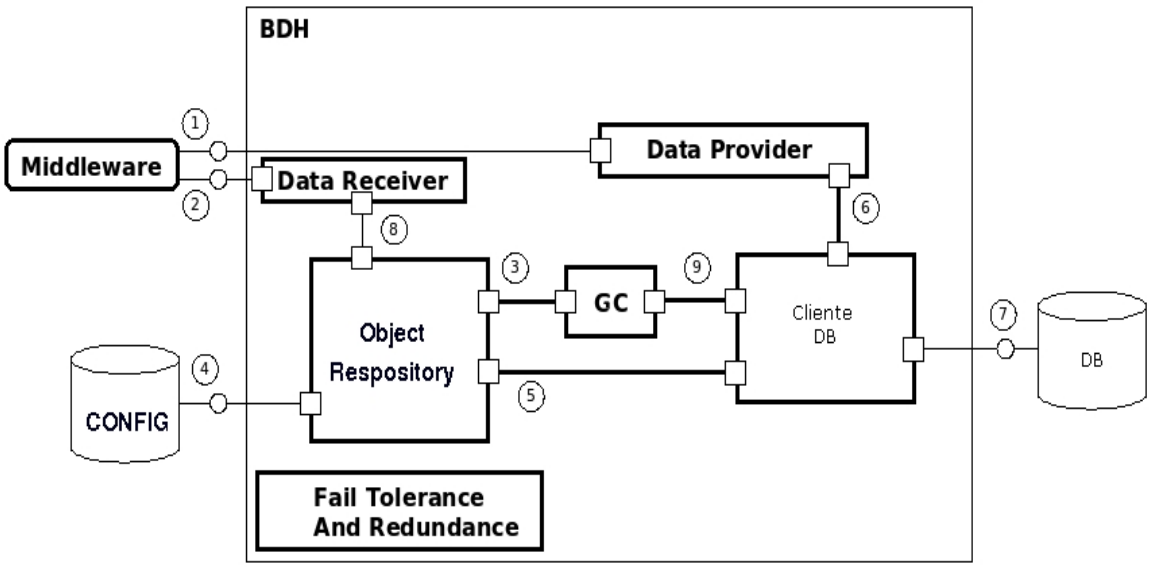

Figura 2: Subsistema BDH.

Data Provider.

Este submódulo es el encargado del envío de información de los servicios históricos hacia todos los módulos externos que la soliciten, a través del *Middleware*.

Subsistemas como el generador de reportes se conectan a través de la interfaz 1, solicitando en forma de consultas SQL (*GTA*) la información de alarmas, eventos, variables y bitácora, y a través de ella misma se entregarán las respuestas.

Cada vez que un cliente solicite una consulta a alguno de los servicios de históricos, el módulo Data Provider se encargará de encolar las solicitudes y de entregar las respuestas una vez las obtenga de los medios persistentes. [12]

Data Receiver.

Posee todas las interfaces de entrada hacia la BDH. Es el encargado de recibir la información de todos los módulos que envíen datos a los históricos.
Sus interfaces se especializan en [13]:

- Recepción de puntos.
- Recepción de alarmas.
- Recepción de eventos.
- Recepción de bitácora.

Object Repository.

La configuración de los históricos es aplicada al módulo a través de la interfaz 4. La configuración de cada uno de los servicios es modelada por objetos en el Object Repository, lo que permite realizar las tareas correspondientes de almacenamiento a los medios persistentes.

Dentro de la configuración de los históricos de variables se destacan los grupos de envío hacia los históricos, los cuales son configurados por los mantenedores en tiempo de edición.

Por otra parte el Object Repository contiene la información de la configuración interna del módulo relativa a tamaños de buffers (*GTA*), tiempos de espera, etc.

Las tareas de mantenimiento de la Base de Datos (BD) también se nutren de las configuraciones modeladas en este submódulo, y utilizan el Garbage Collector (GC), a través de la interfaz 3, para eliminar la información antigua no útil. [14]

Garbage Collector.

Se encarga de las tareas de mantenimiento del historiador dentro de las que se destacan, borrar información de eventos, alarmas, etc. posteriores al día especificado en la configuración, con la finalidad de eliminar información antigua. Dichas tareas las realiza utilizando la interfaz 9, para informarle al ClientDB sobre las acciones requeridas en un instante dado. [15]

ClientDB.

Este submódulo es una capa de abstracción de datos entre los diferentes servicios del historiador y el medio persistente. Por medio de él se realizan las solicitudes de tareas de inserción de nuevos eventos, alarmas, variables y bitácoras, así como todas las tareas de mantenimiento sobre la BD. Además, es el encargado de mantener la lista de conexiones con los clientes que consultan los datos de los diferentes servicios de históricos a través de la interfaz 6. Las conexiones al servicio de BD, se realizan a través de la interfaz 7. [16]

Tolerancia ante fallas.

Es el encargado de interactuar con los módulos de monitoreo para mantener el estado de ejecución de los servicios de históricos.

El Módulo BDH cuenta con 2 niveles fundamentales: el historiador y el servidor de BD. En ambos niveles se requiere de una tolerancia ante fallos que permita un funcionamiento adecuado.

En el nivel del historiador, el sistema de redundancia propuesto es el de *Hot Standby* (*GTA*), con un servidor primario y uno secundario. El servidor secundario sustituirá en sus funciones las del primario, una vez que se detecte que el mismo se encuentra afrontando problemas o errores que le impidan ofrecer el servicio. [17]

Por otra parte, en el nivel servidor de bases de datos, se propone un sistema similar *Hot Standby*, encargado de determinar si el servidor afronta problemas para redirigir las solicitudes a un servidor redundante de BD. [17]

1.5 Programación multi-hilos .

Una "hebra" (*thread*) es un semi-proceso, que posee su propia pila (*stack*), y ejecuta un segmento de código. A diferencia de un proceso real, el *thread* normalmente comparte su memoria con otros *threads* (cuando se tienen varios procesos generalmente se tiene una porción de memoria diferente para cada uno de ellos). Un grupo de *threads* dentro de un mismo proceso comparten la misma memoria, por lo que pueden acceder a las variables globales. Todas estos *threads* se ejecutan en paralelo.

La ventaja de usar un grupo de *threads* sobre el uso de un programa serial normal, es que algunas operaciones pueden ser ejecutadas en paralelo. De este modo, los eventos pueden ser manejados

inmediatamente de su aparición. [18]

La ventaja de usar un grupo de *threads* en vez de un grupo de procesos es que, para el procesador, el cambio entre *threads* es mucho más rápido que si se efectuara entre procesos. Además, la comunicación entre dos *threads* suele ser mucho más rápida y fácil que la comunicación entre dos procesos.

Por otro lado, es importante destacar que por el hecho de que un grupo de *threads* comparten el mismo espacio de memoria, si una de ellas destruye el contenido de esa memoria el resto de los *threads* sufrirían el impacto. Entre procesos, por lo general, el sistema operativo protege un proceso de otro. Así, si un proceso destruye su espacio de memoria, los demás no se verían afectados. Otra ventaja del uso de procesos es que ellos pueden correr en diferentes máquinas, mientras que los *threads* deben correr todos en una misma máquina. [18]

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

Introducción

En el presente capítulo se adquiere una visión práctica del sistema que se pretende desarrollar. En el mismo se expondrán las reglas del negocio, así como los requisitos funcionales y no funcionales que regirán el desarrollo de la solución al problema, definiendo qué esperan los usuarios de la misma. Partiendo de esto se determinarán los casos de uso a realizar. Además se describirán los procesos de las principales funcionalidades del subsistema.

2.1 Soluciones Técnicas.

Para darle cumplimiento al objetivo de este trabajo, se pretende desarrollar el subsistema de comunicaciones para el Módulo BDH que tendrá como funciones, por un lado, la recepción de las estructuras enviadas desde el núcleo de procesamiento de datos -BDTR- a través del *Middleware*, para que sean entregadas al subsistema DataStore. Y por el otro lado, la de gestionar las consultas a la BD, brindando de este modo un acceso controlado a los históricos almacenados.

Para la recepción de datos a través de la red, *Middleware* proporciona un grupo de interfaces que se describen a continuación:

- *InputDataManager*: Interfaz que describe un manejador de datos de entrada. Se usa para recibir alarmas, lotes de alarmas, puntos y lotes de puntos, por la frecuencia tradicional o por una frecuencia especificada. [19]
- *IInputEventManager*: Interfaz que describe un manejador de eventos de entrada. Se usa para recibir eventos. [20]
- *IInputUserLogManager*: Interfaz que describe un manejador de log de usuarios de entrada. Se usa para recibir log de usuarios. [21]
- *IInputCommStatisticsManager*: Interfaz que describe un manejador de estados de comunicación de dispositivos y subcanales. Se usa para recibir los estados.

Así como también provee otro grupo de interfaces para la gestión de consultas a los históricos:

- *QueryClientManager*.
- *QueryServiceManager*.

Los objetos transmitidos a través del *Middleware* tienen una estructura genérica para todos los módulos del SCADA [22]. Debido a esto se hará necesaria la conversión de estos a sus homólogos ya dentro de BDH, para de esta manera optimizar el flujo interno, procesamiento y almacenamiento de los mismos.

La conexión a la BD dentro del Módulo BDH se gestiona usando la biblioteca de acceso a datos “pqxx” que brinda un grupo de funcionalidades y estructuras para el manejo de las consultas. Entre ellas se encuentra la clase “pqxx::result”, contenedora del resultado de una consulta ejecutada, que ofrece un acceso amigable a los datos. Para abstraer a los módulos que soliciten consultas a los históricos del uso de “pqxx”, se desarrollará una estructura similar a “pqxx::result” que ofrecerá sus principales funcionalidades, pero no será dependiente de esta biblioteca.

Los resultados de las consultas a la BD se transmitirán del servidor al cliente con la siguiente estructura:

- *HEADER:* Contendrá la información correspondiente a los registros devueltos por la ejecución de la consulta, entiéndase por esto nombre y tipos de los campos.
- *BODY*: Bloque contenedor de datos.
- *FOOTER*: Bloque final. Indica el fin del resultado de una solicitud ejecutada. También puede contener datos.
- *ERROR*: Bloque especial que será enviado en caso de ocurrir algún error de conexión con la BD, o de que el servidor esté totalmente ocupado en ese momento. Contiene además la descripción del error.

De este modo se facilitará la interpretación de las mismas del lado del cliente.

Debido a la necesidad de agilizar el proceso de recepción de datos y el acceso a los históricos, se hace necesario la utilización de la programación multi-hilos. El subsistema se ejecutará como dos hilos de procesamiento: uno para la recepción de información y otro para la tarea de proveer acceso a los datos históricos almacenados en la BD. Para ello se va a utilizar la biblioteca *OpenThreads* que, además de ser libre, ofrece un manejo amigable de los hilos de procesamiento y secciones críticas

(*GTA*). Por otro lado, cabe destacar que es una biblioteca multiplataforma (*GTA*), por lo que sería muy fácil la migración del subsistema a *Windows*. [23]

2.2 Modelo de Dominio.

En la figura 3 se muestra el modelo del dominio, en el mismo se representan los principales conceptos que serán manejados para alcanzar los objetivos trazados para este proyecto.

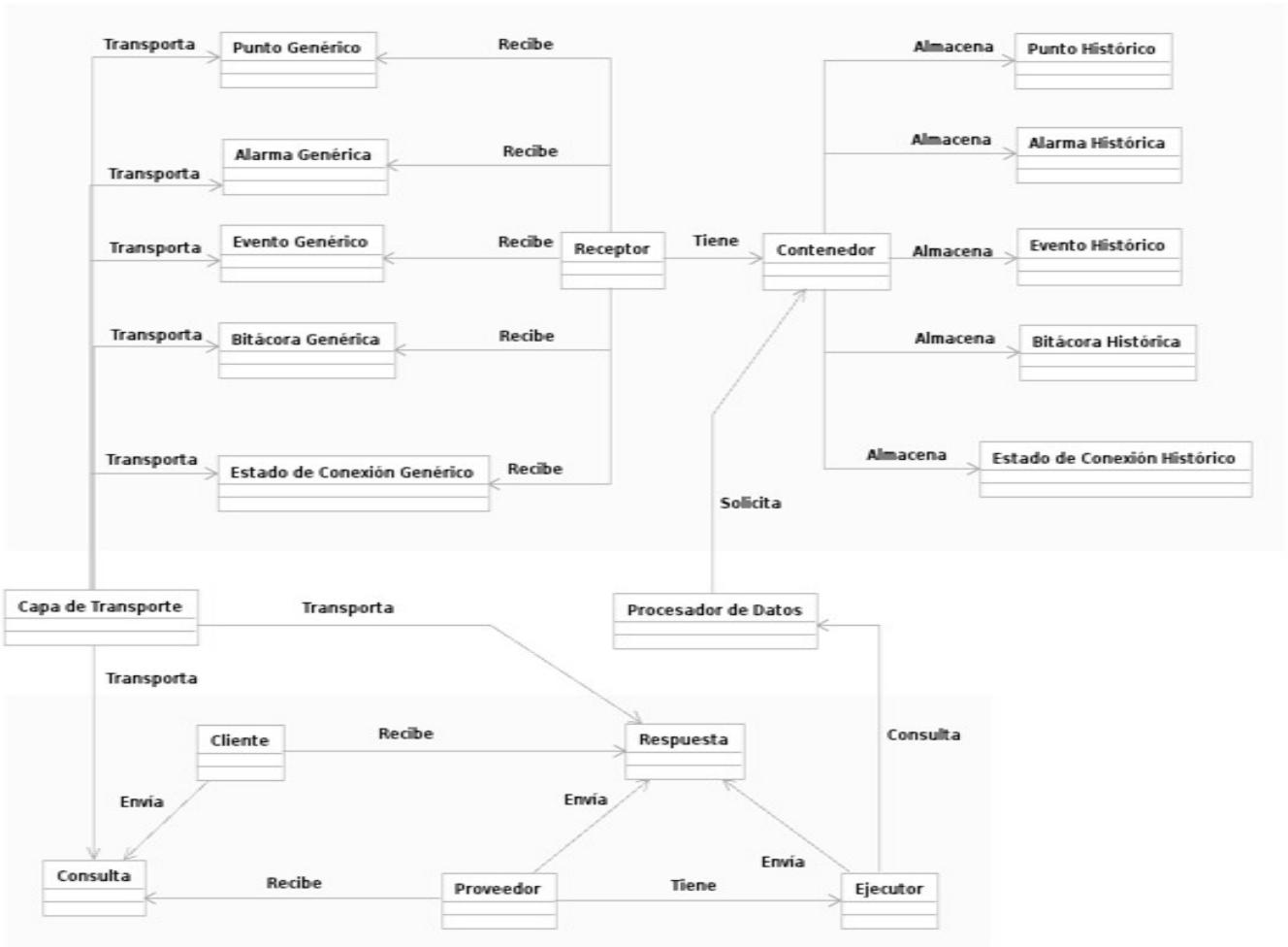

Figura 3: Diagrama de Dominio.

2.2.1 Glosario de términos del dominio.

Capa de Transporte: Módulo encargado de la transferencia de datos (*Middleware*).

Punto Genérico: Estructura genérica de una variable.

Alarma Genérica: Estructura genérica de una alarma.

Evento Genérico: Estructura genérica de un evento.

Bitácora Genérica: Estructura genérica de una bitácora.

Estado de Conexión Genérico: Estructura genérica de un estado de conexión.

Receptor: Encargado de recibir los datos transmitidos a través de la capa de transporte.

Contenedor: Convierte los datos genéricos recibidos y los transforma a datos históricos para luego almacenarlos temporalmente hasta ser solicitados por el procesador de datos.

Punto Histórico: Estructura de una variable histórica.

Alarma Histórica: Estructura de una alarma histórica.

Evento Histórico: Estructura de un evento histórico.

Bitácora Histórica: Estructura de una bitácora histórica.

Estado de Conexión Histórico: Estructura de un estado de conexión histórico.

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. Controla el número de consultas en ejecución concurrente. 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.

Consulta: Solicitud de datos de los históricos. Ejemplo: Consulta *SQL (Structured Query Lenguaje)*

Respuesta: Resultado de la ejecución de una consulta o mensaje de error.

Procesador de Datos: Contiene la lógica de almacenamiento y acceso a los datos históricos que persisten en la BD.

2.3 Reglas del negocio.

- La recepción y transmisión de datos se llevará a cabo a través del *Middleware*.
- El número de solicitudes de datos en ejecución de modo concurrente será configurado en el momento de iniciar el subsistema.
- El subsistema debe soportar hasta 50 consultas concurrentes a la BD.
- El subsistema debe ser capaz de recibir el volumen de datos transmitidos por *Middleware* sin pérdidas de información.

2.4 Requerimientos del sistema.

A continuación se exponen los requisitos funcionales, que no son más que las capacidades o condiciones que el sistema debe cumplir, y los requisitos no funcionales, o sea, propiedades o cualidades que el producto debe tener.

2.4.1 Requisitos funcionales. [24]

1. Recibir datos a través del *Middleware*.
   - 1.1. Recibir puntos.
     - 1.1.1. Transformar los puntos genéricos recibidos a puntos históricos.
     - 1.1.2. Almacenar de modo temporal los puntos recibidos.
   - 1.2. Recibir alarmas.
     - 1.2.1. Transformar las alarmas genéricas recibidas a alarmas históricas.
     - 1.2.2. Almacenar de modo temporal las alarmas recibidas.
   - 1.3. Recibir eventos.
     - 1.3.1. Transformar los eventos genéricos recibidos a eventos históricos.

1.3.2. Almacenar de modo temporal los eventos recibidos.

1.4. Recibir bitácoras.

1.4.1. Transformar las bitácoras genéricas recibidas a bitácoras históricas.

1.4.2. Almacenar de modo temporal las bitácoras recibidas.

1.5. Recibir estados de conexión.

1.5.1. Transformar los estados de conexión genéricos recibidos a estados de conexión históricos.

1.5.2. Almacenar de modo temporal los estados de conexión recibidos.

2. Registrar el cliente solicitante de datos históricos en el servidor.
   2.1. Enviar solicitud de registro al servidor.
   2.2. Recibir confirmación de registro enviada por el servidor.
   2.3. Manejar por parte del servidor la lista de clientes registrados.
3. Gestionar solicitud de datos.
   3.1. Enviar solicitud de datos al servidor.
   3.2. Poner en ejecución una solicitud de datos.
   3.3. Devolver la(s) respuesta(s).
   3.4. Recibir la(s) respuesta(s).
      3.4.1. Interpretar la(s) respuesta(s) recibida(s).
   3.5. Debe manejar hasta 50 consultas concurrentes a la BD.

2.4.2 Requisitos no funcionales.

Software

- Sistema operativo GNU/Linux, distribución Debian, Kernel 2.6

Diseño e implementación

- Lenguaje de programación C++.
- Se regirá por la filosofía de Programación Orientada a Objetos.

Rendimiento

- Haga uso óptimo de la memoria.

Portabilidad

- Multiplataforma.

Ayuda y documentación

- Va a contar con Ayuda y especificaciones de uso en línea.

2.5 Diagrama de Casos de Uso.

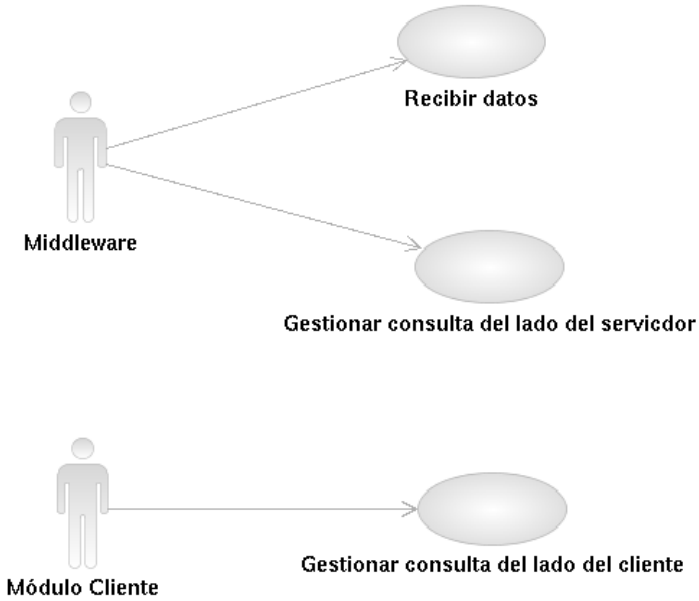

Figura 4: Diagrama de Casos de Uso.

2.6 Expansión de los Casos de Uso.

2.6.1 Definición de los actores.

Tabla 2: Definición de los actores.

Actores	**Justificación**
Middleware	Capa de transporte encargada de entregar los datos publicados por la BDTR, iniciando el proceso de recepción para su posterior almacenamiento.
Módulo Cliente	Módulos que solicitan información de los históricos.

2.6.2 Listado de casos de uso.

Tabla 3: Caso de uso Recibir datos.

CU-1	**Recibir datos**
Actor	*Middleware*
Descripción	Recibir los datos genéricos publicados por la BDTR, los cuales son transmitidos a través de la capa de transporte. Una vez que éstos son recibidos son trasformados a datos históricos y almacenados temporalmente hasta ser solicitados para su procesamiento.
Referencia	1, 1.1, 1.1.1, 1.1.2, 1.2, 1.2.1, 1.2.2, 1.3, 1.3.1, 1.3.2, 1.4, 1.4.1, 1.4.2, 1.5, 1.5.1, 1.5.2

Tabla 4: Caso de uso Gestionar consulta del lado del cliente.

CU-2	**Gestionar consulta del lado del cliente**
Actor	Módulo Cliente
Descripción	El cliente se registra en el servidor de históricos, envía una solicitud de datos históricos e interpreta la respuesta recibida.
Referencia	2, 2.1, 2.2, 3, 3.1, 3.4, 3.4.1

Tabla 5: Caso de uso Gestionar consulta del lado del servidor.

CU-3	**Gestionar consulta del lado del servidor**
Actor	*Middleware*
Descripción	Registra a los clientes que lo soliciten. Recibe una solucitud de consulta, la valida y la pone en

	ejecución. Se devuelve(n) la(s) respuesta(s) de la ejecución de la consulta o mensaje(s) de error en caso de ser necesario.
Referencia	2, 2.3, 3, 3.2, 3.3, 3.5

2.6.3 Casos de uso por ciclo.

Tabla 6: Casos de uso por ciclo.

Cód	Nombre de caso de uso	Paquete	Justificación de la selección.
CU-1	Recibir datos.	1	
CU-2	Gestionar consulta del lado del cliente.	1	
CU-3	Gestionar consulta del lado del servidor.	1	

2.6.4 Casos de uso expandidos.

Tabla 7: Expansión del caso de uso Recibir datos.

Caso de uso	
CU-1	Recibir datos.
Propósito	Recibir los datos genéricos transmitidos a través del *Middleware* y almacenarlos temporalmente para luego ser procesados.
Actores: *Middleware*	
Resumen: Se inicia el caso de uso cuando un dato genérico llega a través del *Middleware*, éste es recibido, convertido en un dato histórico, y almacenado temporalmente hasta que sea solicitado para su procesamiento y posterior almacenamiento en la BD.	
Referencias	1, 1.1, 1.1.1, 1.1.2, 1.2, 1.2.1, 1.2.2, 1.3, 1.3.1, 1.3.2, 1.4, 1.4.1, 1.4.2, 1.5, 1.5.1, 1.5.2
Acción del actor	**Respuesta del sistema**
Sección: Recibir punto.	

1- Llega un punto genérico.	1.1- Se recibe el punto genérico.
	1.2- Se trasforma a un punto histórico.
	1.3- Se almacena el punto histórico en un *buffer* que podrá ser accedido por el núcleo de BDH para su procesamiento y almacenamiento.
Sección: Recibir alarma.	
1- Llega una alarma genérica.	1.1- Se recibe la alarma genérica.
	1.2- Se trasforma a una alarma histórica.
	1.3- Se almacena la alarma histórica en un *buffer* que podrá ser accedida por el núcleo de BDH para su procesamiento y almacenamiento.
Sección: Recibir evento.	
1- Llega un evento genérico.	1.1- Se recibe el evento genérico.
	1.2- Se trasforma a un evento histórico.
	1.3- Se almacena el evento histórico en un *buffer* que podrá ser accedido por el núcleo de BDH para su almacenamiento.
Sección: Recibir bitácora.	
1- Llega una bitácora genérica.	1.1- Se recibe la bitácora genérica.
	1.2- Se trasforma a una bitácora histórica.
	1.3- Se almacena la bitácora histórica en un *buffer* que podrá ser accedida por el núcleo de BDH para su almacenamiento.
Sección: Recibir estado de conexión.	
1- Llega un estado de conexión genérico.	1.1- Se recibe el estado de conexión genérico.
	1.2- Se trasforma a un estado de conexión histórico.
	1.3- Se almacena el estado de conexión histórico en un *buffer* que podrá ser accedido por el núcleo de

	BDH para su procesamiento y almacenamiento.
Precondiciones	■ Los servicios del *Middleware* deben estar corriendo.
Poscondiciones	■ Los datos históricos están listos para ser procesados y almacenados en la BD.

Tabla 8: Expansión del caso de uso Gestionar consulta del lado del cliente.

Caso de uso	
CU-2	Gestionar consulta del lado del cliente.
Propósito	Manejar las solicitudes de datos históricos desde los módulos que lo requieran.
Actores: Módulo Cliente	
Resumen: El caso de uso se inicia cuando un módulo cliente necesita consultar datos históricos, para lo cual se registra en el servidor, y luego envía una solicitud de datos a éste, para finalmente interpretar la(s) respuesta(s) recibida(s).	
Referencias	2, 2.1, 2.2, 3, 3.1, 3.4, 3.4.1
Acción del actor	**Respuesta del sistema**
Sección: Registrar cliente.	
1- Crea un cliente.	1.1- Envía una solicitud de registro al servidor.
	1.2- Recibe el identificador asignado al cliente.
Sección: Enviar solicitud de datos históricos	
1- Crea la solicitud.	1.1- Envía la solicitud de datos al servidor.
Sección: Recibir respuesta(s) de solicitud de datos históricos.	
	1- Se recibe el bloque de información enviado por el servidor.

	1.1- Se interpreta para identificar si es de tipo *HEADER*, *BODY*, *FOOTER* o *ERROR*.
	1.2- En cualquiera de los tres primeros casos se procesan los datos e insertan en el contenedor del cliente. En caso de ser *ERROR* se le informa al módulo cliente.
Precondiciones	■ Los servicios del *Middleware* deben estar corriendo. ■ El Módulo BDH debe estar en ejecución.
Poscondiciones	■ El cliente se mantiene registrado.

Tabla 9: Expansión del caso de uso Gestionar consulta del lado del servidor.

Caso de uso	
CU-3	Gestionar consulta del lado del servidor.
Propósito	Manejar las solicitudes de datos históricos en el servidor.
Actores: *Middleware*	
Resumen: Este caso de uso se inicia cuando llega al servidor una solicitud de registro. Los clientes una vez registrados envían solicitudes de datos que son gestionadas en el servidor para luego ser puestas en ejecución. La(s) respuesta(s) de éstas son enviadas a los clientes a través de *Middleware*.	
Referencias	2, 2.3, 3, 3.2, 3.3, 3.5

Acción del actor	**Respuesta del sistema**
Sección: Registrar cliente	
1- Llega una solicitud de registro.	1.1- Se recibe la solicitud.
	1.2- Se asigna un identificador al cliente solicitante y se almacena el registro.

2- Transmite el identificador al cliente.	1.3- Se envía al cliente su identificador.
Sección: Recibir solicitud de datos históricos	
Curso Normal de los Eventos	
1- Llega una solicitud de datos históricos.	1.1- Se recibe la solicitud y se valida el identificador del cliente.
	1.2- Se verifica que el servidor no esté totalmente ocupado (ejecutando el número máximo de consultas concurrentes a la BD).
	1.3- Se pone en ejecución la solicitud de datos.
2- Se transmiten los resultados al cliente.	1.4- Se devuelven los resultados.
Cursos Alternos	
1.1- En caso de no estar registrado se envía un mensaje de error que se transmite al cliente a través del *Middleware*. 1.2-En caso de estar ocupado se envía un mensaje al cliente a través del *Middleware*.	
Precondiciones	■ Los servicios del *Middleware* deben estar corriendo.
Poscondiciones	■ Los clientes recibieron la información solicitada o el mensaje de algún posible error.

Consideraciones finales

Con el desarrollo de este capítulo han sido sentadas las bases técnicas por las que se va a regir el sistema. Se ha definido exactamente qué espera el usuario del mismo a través de las reglas del negocio, los requisitos no funcionales y los funcionales, recogidos estos últimos en tres grupos:

- Recibir datos a través de *Middleware*.
- Registrar el cliente solicitante de datos históricos en el servidor.
- Gestionar solicitud de datos.

Tras el análisis de todos ellos se determinaron y fueron descritos tres casos de uso principales:

- Recibir datos.
- Gestionar consulta del lado del cliente.
- Gestionar consulta del lado del servidor.

Partiendo de todo esto, en el siguiente capítulo se diseñará e implementará una estructura de clases en concordancia con las técnicas citadas para de esta forma darle cumplimiento a todos los objetivos de este proyecto.

CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN

Introducción

En este capítulo se diseñará un sistema de clases, en correspondencia con las técnicas de la programación orientada a objetos, que luego serán implementadas teniendo como resultado final un subsistema funcional que cumpla con los requisitos del usuario. En él se expondrán los diagramas de clases de diseño, separados en los tres paquetes que componen el subsistema para un mejor entendimiento de los mismos. Estos diagramas brindan una representación muy completa de todo el subsistema, mostrando las clases, métodos y las relaciones entre ellos. Además, como parte de la transición de la fase de diseño a la de implementación, se presentarán los diagramas de componentes de cada uno de los paquetes.

3.1 Diagrama de clases paquete *DataReceiver*.

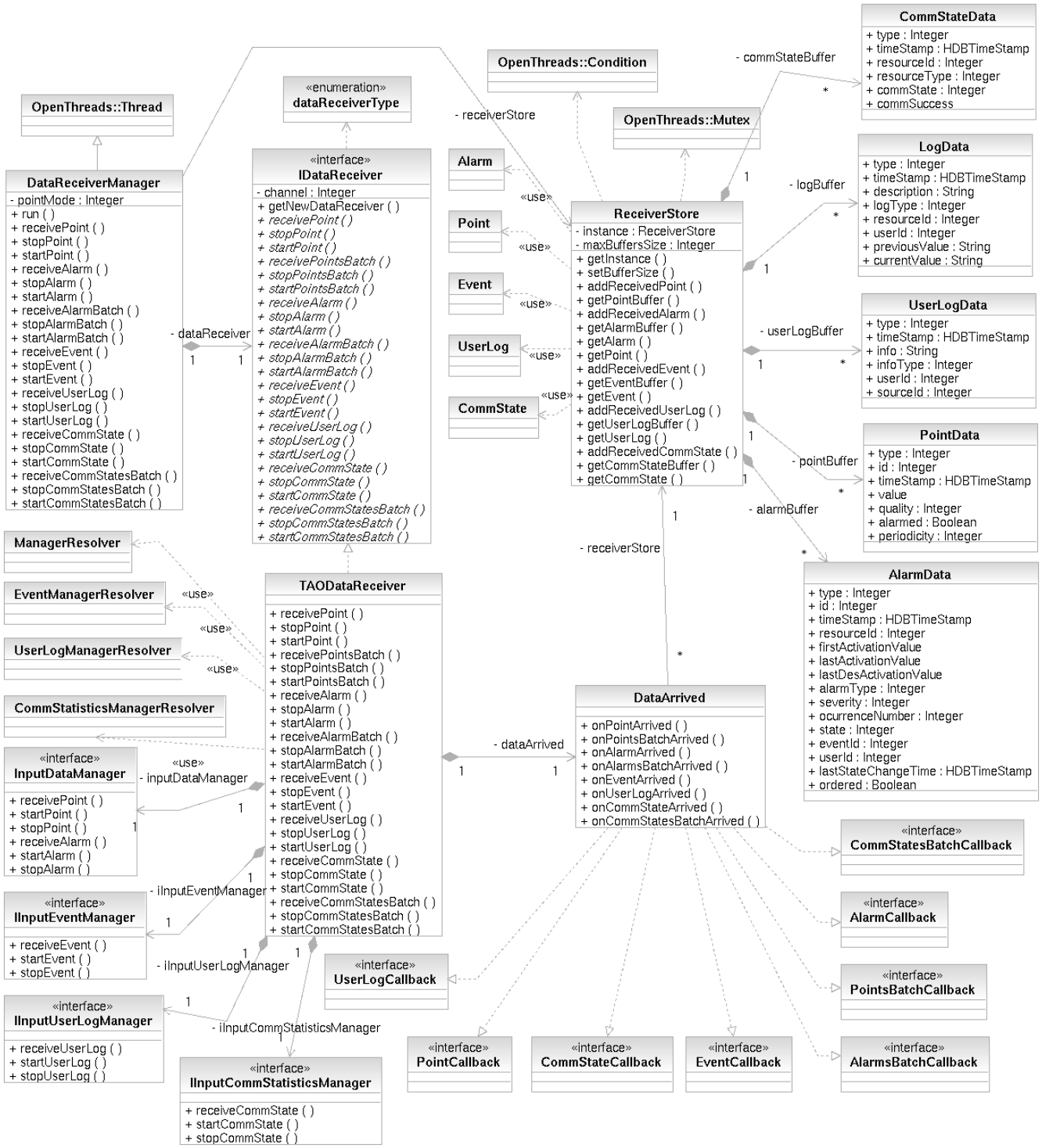

Figura 5: Diagrama de clases de diseño del paquete DataReceiver.

3.1.1 Descripción de las clases del paquete *DataReceiver*.

Tabla 10: Descripción de la clase DataReceiverManager.

Nombre: DataReceiverManager		
Tipo de clase	Controladora	
Atributo		**Tipo**
dataReceiver		IDataReceiver
pointMode		unsigned short int
Para cada responsabilidad:		
Nombre:	run()	
Descripción:	Reimplementación del método *run* de la clase OpenThreads::Thread, habilita la recepción de todas las estructuras.	
Nombre:	receivePoint()	
Descripción:	Inicia la recepción de puntos, teniendo en cuenta el modo establecido.	
Nombre:	stopPoint()	
Descripción:	Inhabilita la recepción de puntos, teniendo en cuenta el modo establecido.	
Nombre:	startPoint()	
Descripción:	Habilita la recepción de puntos, teniendo en cuenta el modo establecido.	
Nombre:	receiveAlarm()	
Descripción:	Inicia la recepción de alarmas.	
Nombre:	stopAlarm()	
Descripción:	Inhabilita la recepción de alarmas.	
Nombre:	startAlarm()	
Descripción:	Habilita la recepción de alarmas.	
Nombre:	receiveAlarmBatch()	
Descripción:	Inicia todo el proceso de recepción de *buffers* de alarmas.	
Nombre:	stopAlarmBatch()	

Descripción:	Inhabilita recepción la de *buffers* de alarmas.
Nombre:	startAlarmBatch()
Descripción:	Habilita la recepción de *buffers* de alarmas.
Nombre:	receiveEvent()
Descripción:	Inicia la recepción de eventos.
Nombre:	stopEvent()
Descripción:	Inhabilita la recepción de eventos.
Nombre:	startEvent()
Descripción:	Habilita la recepción de eventos.
Nombre:	receiveUserLog()
Descripción:	Inicia la recepción de bitácoras.
Nombre:	stopUserLog()
Descripción:	Inhabilita la recepción de bitácoras.
Nombre:	startUserLog()
Descripción:	Habilita la recepción de bitácoras.
Nombre:	receiveCommState()
Descripción:	Inicia la recepción de estados de conexión.
Nombre:	stopCommState()
Descripción:	Inhabilita la recepción de estados de conexión.
Nombre:	startCommState()
Descripción:	Habilita la recepción de estados de conexión.
Nombre:	receiveCommStatesBatch()
Descripción:	Inicia la recepción de *buffers* de estados de conexión.
Nombre:	stopCommStatesBatch()
Descripción:	Inhabilita la recepción de *buffers* de estados de conexión.
Nombre:	startCommStatesBatch()

Descripción:	Habilita la recepción de *buffers* de estados de conexión.

Tabla 11: Descripción de la clase IDataReceiver.

Nombre: IDataReceiver		
Tipo de clase	Controladora	
Atributo		**Tipo**
channel		int
Para cada responsabilidad:		
Nombre:	getNewDataReceiver(argDataReceiverType: dataReceiverType,argc: int, argv: char**, channel: int)	
Descripción:	Construye una instancia DataReceiver según el tipo pasado por parámetros.	
Nombre:	receivePoint()	
Descripción:	Inicia la recepción de puntos.	
Nombre:	stopPoint()	
Descripción:	Inhabilita la recepción de puntos.	
Nombre:	startPoint()	
Descripción:	Habilita la recepción de puntos.	
Nombre:	receivePointsBatch()	
Descripción:	Inicia la recepción de *buffers* de puntos.	
Nombre:	stopPointsBatch()	
Descripción:	Inhabilita la recepción de *buffers* de puntos.	
Nombre:	startPointsBatch()	
Descripción:	Habilita la recepción de *buffers* de puntos.	
Nombre:	receiveAlarm()	

Descripción:	Inicia la recepción de alarmas.
Nombre:	stopAlarm()
Descripción:	Inhabilita la recepción de alarmas.
Nombre:	startAlarm()
Descripción:	Habilita la recepción de alarmas.
Nombre:	receiveAlarmBatch()
Descripción:	Inicia todo el proceso de recepción de *buffers* de alarmas.
Nombre:	stopAlarmBatch()
Descripción:	Inhabilita recepción la de *buffers* de alarmas.
Nombre:	startAlarmBatch()
Descripción:	Habilita la recepción de *buffers* de alarmas.
Nombre:	receiveEvent()
Descripción:	Inicia la recepción de eventos.
Nombre:	stopEvent()
Descripción:	Inhabilita la recepción de eventos.
Nombre:	startEvent()
Descripción:	Habilita la recepción de eventos.
Nombre:	receiveUserLog()
Descripción:	Inicia la recepción de bitácoras.
Nombre:	stopUserLog()
Descripción:	Inhabilita la recepción de bitácoras.
Nombre:	startUserLog()
Descripción:	Habilita la recepción de bitácoras.
Nombre:	receiveCommState()
Descripción:	Inicia la recepción de estados de conexión.
Nombre:	stopCommState()

Descripción:	Inhabilita la recepción de estados de conexión.
Nombre:	startCommState()
Descripción:	Habilita la recepción de estados de conexión.
Nombre:	receiveCommStatesBatch()
Descripción:	Inicia la recepción de *buffers* de estados de conexión.
Nombre:	stopCommStatesBatch()
Descripción:	Inhabilita la recepción de *buffers* de estados de conexión.
Nombre:	startCommStatesBatch()
Descripción:	Habilita la recepción de *buffers* de estados de conexión.

Tabla 12: Descripción de la clase TAODataReceiver.

Nombre: TAODataReceiver		
Tipo de clase	Controladora	
Atributo		**Tipo**
inputDataManager		InputDataManager
iInputEventManager		IInputEventManager
iInputUserLogManager		IInputUserLogManager
iInputCommStatisticsManager		IInputCommStatisticsManager
dataArrived		DataArrived
Para cada responsabilidad:		
Nombre:	receivePoint()	
Descripción:	Inicia la recepción de puntos.	
Nombre:	stopPoint()	

Descripción:	Inhabilita la recepción de puntos.
Nombre:	startPoint()
Descripción:	Habilita la recepción de puntos.
Nombre:	receivePointsBatch()
Descripción:	Inicia la recepción de *buffers* de puntos.
Nombre:	stopPointsBatch()
Descripción:	Inhabilita la recepción de *buffers* de puntos.
Nombre:	startPointsBatch()
Descripción:	Habilita la recepción de *buffers* de puntos.
Nombre:	receiveAlarm()
Descripción:	Inicia la recepción de alarmas.
Nombre:	stopAlarm()
Descripción:	Inhabilita la recepción de alarmas.
Nombre:	startAlarm()
Descripción:	Habilita la recepción de alarmas.
Nombre:	receiveAlarmBatch()
Descripción:	Inicia todo el proceso de recepción de *buffers* de alarmas.
Nombre:	stopAlarmBatch()
Descripción:	Inhabilita recepción la de *buffers* de alarmas.
Nombre:	startAlarmBatch()
Descripción:	Habilita la recepción de *buffers* de alarmas.
Nombre:	receiveEvent()
Descripción:	Inicia la recepción de eventos.
Nombre:	stopEvent()
Descripción:	Inhabilita la recepción de eventos.
Nombre:	startEvent()

Descripción:	Habilita la recepción de eventos.
Nombre:	receiveUserLog()
Descripción:	Inicia la recepción de bitácoras.
Nombre:	stopUserLog()
Descripción:	Inhabilita la recepción de bitácoras.
Nombre:	startUserLog()
Descripción:	Habilita la recepción de bitácoras.
Nombre:	receiveCommState()
Descripción:	Inicia la recepción de estados de conexión.
Nombre:	stopCommState()
Descripción:	Inhabilita la recepción de estados de conexión.
Nombre:	startCommState()
Descripción:	Habilita la recepción de estados de conexión.
Nombre:	receiveCommStatesBatch()
Descripción:	Inicia la recepción de *buffers* de estados de conexión.
Nombre:	stopCommStatesBatch()
Descripción:	Inhabilita la recepción de *buffers* de estados de conexión.
Nombre:	startCommStatesBatch()
Descripción:	Habilita la recepción de *buffers* de estados de conexión.

Tabla 13: Descripción de la clase DataArrived.

Nombre: DataArrived	
Tipo de clase	Controladora
Atributo	**Tipo**

Para cada responsabilidad:	
Nombre:	onPointArrived(point: Point)
Descripción:	Entrega el punto recibido a la clase ReceiverStore.
Nombre:	onPointsBatchArrived(pointsBatch: PointsBatch)
Descripción:	Entrega cada uno de los puntos contenidos en el *buffer* recibido a la clase ReceiverStore.
Nombre:	onAlarmArrived(alarm: Alarm)
Descripción:	Entrega la alarma recibida a la clase ReceiverStore.
Nombre:	onAlarmsBatchArrived(alarmsBatch: AlarmsBatch)
Descripción:	Entrega cada uno de las alarmas contenidas en el *buffer* recibido a la clase ReceiverStore.
Nombre:	onEventArrived(event: Event)
Descripción:	Entrega el evento recibido a la clase ReceiverStore.
Nombre:	onUserLogArrived(userLog: UserLog)
Descripción:	Entrega la bitácora recibida a la clase ReceiverStore.
Nombre:	onCommStateArrived(commState: CommState)
Descripción:	Entrega el estado de conexión recibido a la clase ReceiverStore.
Nombre:	onCommStatesBatchArrived(commStatesBatch: CommStatesBatch)
Descripción:	Entrega cada uno de los estados de conexión contenidos en el *buffer* recibido a la clase ReceiverStore.

Tabla 14: Descripción de la clase ReceiverStore.

Nombre: ReceiverStore		
Tipo de clase	Controladora	
Atributo		**Tipo**
pointBuffer		vector<PointData* >
alarmBuffer		vector<AlarmData* >
eventBuffer		vector<LogData* >
userLogBuffer		vector<UserLogData* >
commStateBuffer		vector<CommStateData* >
instance		ReceiverStore
maxBuffersSize		unsigned int
Para cada responsabilidad:		
Nombre:	addReceivedPoint(argPoint: Point)	
Descripción:	Adiciona un punto al *buffer* que almacena esta estructura. En caso de haber alcanzado el tamaño máximo desecha la primera estructura almacenada en el *buffer*.	
Nombre:	getPointBuffer()	
Descripción:	Da acceso al *buffer* que almacena la estructura PointData.	
Nombre:	getPoint()	
Descripción:	Devuelve el primer elemento del *buffer* de puntos.En caso de estar vacío, retorna NULL.	
Nombre:	addReceivedAlarm(argAlarm: Alarm)	
Descripción:	Adiciona una alarma al *buffer* que almacena esta estructura. En caso de haber alcanzado el tamaño máximo desecha la primera estructura almacenada en el *buffer*.	
Nombre:	getAlarmBuffer()	
Descripción:	Da acceso al *buffer* que almacena la estructura AlarmData.	

Nombre:	getAlarm()
Descripción:	Devuelve el primer elemento del *buffer* de alarmas. En caso de estar vacío, retorna NULL.
Nombre:	addReceivedEvent(argEvent: Event)
Descripción:	Adiciona un evento al *buffer* que almacena esta estructura. En caso de haber alcanzado el tamaño máximo desecha la primera estructura almacenada en el *buffer*.
Nombre:	getEventBuffer()
Descripción:	Da acceso al *buffer* que almacena la estructura LogData.
Nombre:	getEvent()
Descripción:	Devuelve el primer elemento del *buffer* de eventos.En caso de estar vacío, retorna NULL.
Nombre:	addReceivedUserLog(argUserLog: UserLog)
Descripción:	Adiciona una bitácora al *buffer* que almacena esta estructura. En caso de haber alcanzado el tamaño máximo desecha la primera estructura almacenada en el *buffer*.
Nombre:	getUserLogBuffer()
Descripción:	Da acceso al *buffer* que almacena la estructura UserLogData.
Nombre:	getUserLog()
Descripción:	Devuelve el primer elemento del *buffer* de bitácoras.En caso de estar vacío, retorna NULL.
Nombre:	addReceivedCommState(argCommState: CommState)
Descripción:	Adiciona un estado de conexión al *buffer* que almacena esta estructura. En caso de haber alcanzado el tamaño máximo desecha la primera estructura almacenada en el *buffer*.
Nombre:	getCommStateBuffer()
Descripción:	Da acceso al *buffer* que almacena la estructura CommStateData.
Nombre:	getCommState()

Descripción:	Devuelve el primer elemento del *buffer* de estados de conexión.En caso de estar vacío, retorna NULL.

3.2 Diagrama de clases paquete *DataProvider*.

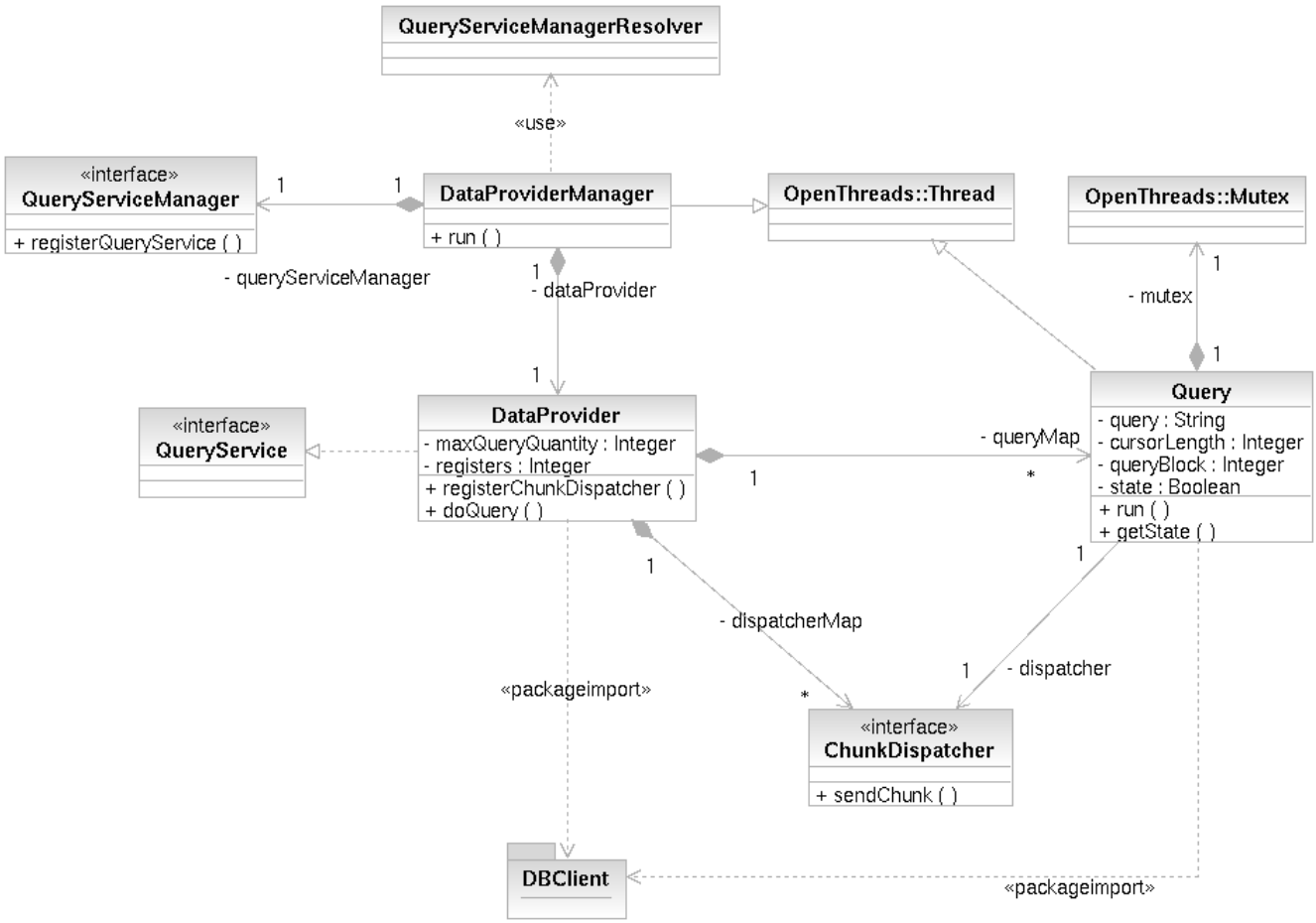

Figura 6: Diagrama de clases de diseño del paquete DataProvider.

3.2.1 Descripción de las clases del paquete *DataProvider*.

Tabla 15: Descripción de la clase DataProviderManager.

Nombre: DataProviderManager	
Tipo de clase	Controladora
Atributo	**Tipo**
queryServiceManager	QueryServiceManager
dataProvider	DataProvider

Para cada responsabilidad:	
Nombre:	run()
Descripción:	Implementación del método run de la clase Thread. Pone en ejecucion el DataProvider.

Tabla 16: Descripción de la clase DataProvider.

Nombre: DataProvider		
Tipo de clase	Controladora	
Atributo		**Tipo**
dispatcherMap		map<unsigned int,ChunkDispatcher>
queryMap		map<unsigned int, Query>
maxQueryQuantity		unsigned int
registers		unsigned short
dbClient		DBClient
Para cada responsabilidad:		
Nombre:	doQuery(query: string, maxResultSize: short int, maxChunkSize: short int, clientID: short int)	
Descripción:	Pone en ejecución una consulta solucitada por un cliente. Valida el número de consultas en ejecución.	
Nombre:	registerChunkDispatcher(chunkDispatcher: ChunkDispatcher)	
Descripción:	Registra clientes.	

Tabla 17: Descripción de la clase Query.

Nombre: Query		
Tipo de clase	Controladora	
Atributo		**Tipo**
dispatcher		ChunkDispatcher
query		string
cursorLength		unsigned int
queryBlock		unsigned int
dbClient		DBClient
state		bool
mutex		Mutex
Para cada responsabilidad:		
Nombre:	run()	
Descripción:	Implementación del método run de la clase Thread. Pone en ejecución la consulta.	

3.3 Diagrama de clases paquete *DataQuery*.

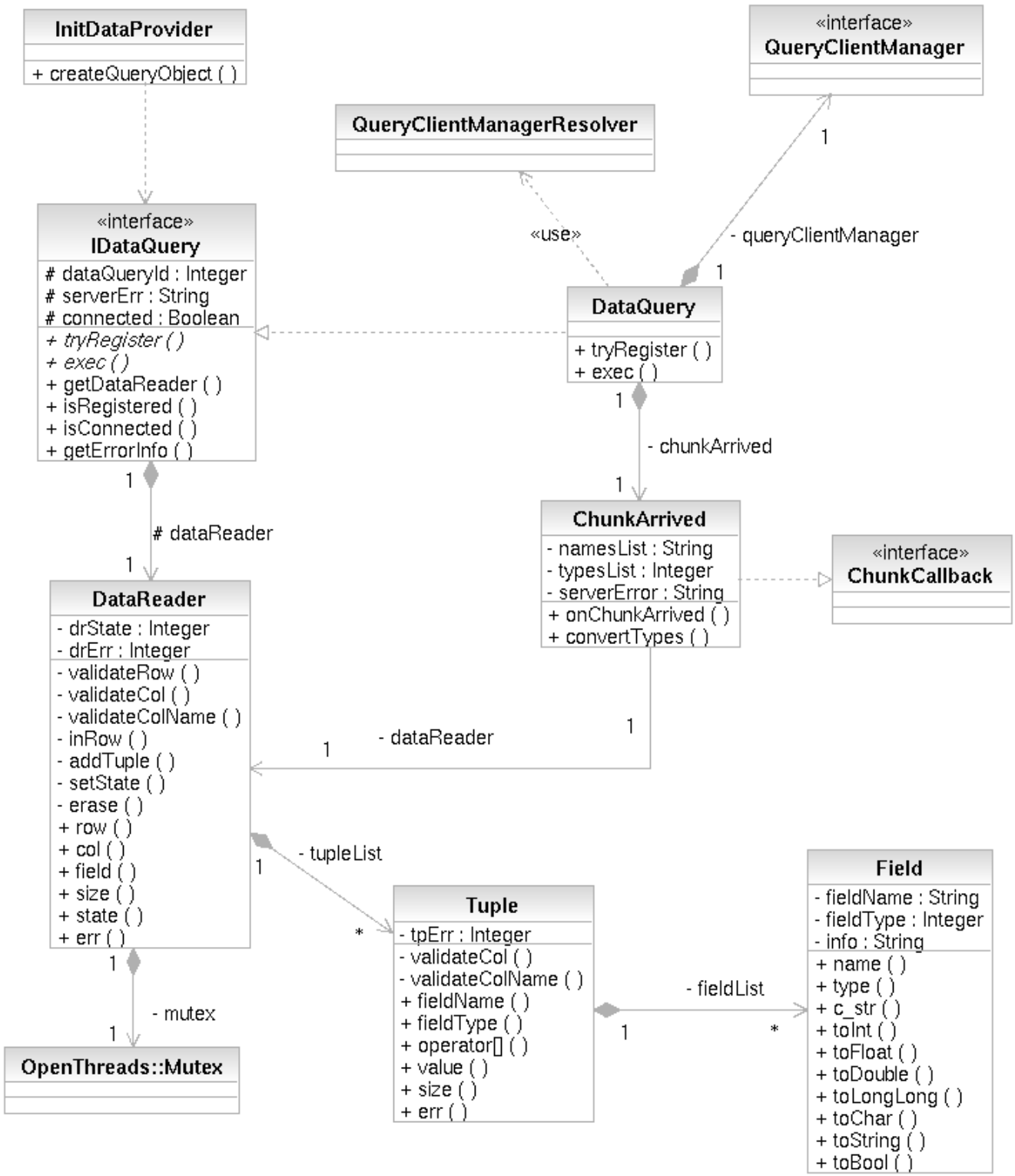

Figura 7: Diagrama de clases de diseño del paquete DataQuery.

3.3.1 Descripción de las clases del paquete *DataQuery*.

Tabla 18: Descripción de la clase InitDataProvider.

Nombre: InitDataProvider	
Tipo de clase	Controladora
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	createQueryObject(argc: int,argv: char**)
Descripción:	Crea una nueva instancia de la clase DataQuery.

Tabla 19: Descripción de la clase IDataQuery.

Nombre: IDataQuery	
Tipo de clase	Controladora
Atributo	**Tipo**
dataReader	DataReader
dataQueryId	unsigned int
serverErr	string
connected	bool
Para cada responsabilidad:	
Nombre:	tryRegister()
Descripción:	Reintenta registrar el cliente en el servidor.
Nombre:	exec(query: string, block: unsigned int, cursorLength: unsigned int)
Descripción:	Envia una consulta de datos al servidor.
Nombre:	getDataReader()
Descripción:	Devuelve una referencia al DataReader.

Nombre:	isRegistered()
Descripción:	Devuelve *true* si el cliente está registrado, *false* en caso contrario.
Nombre:	isConnected()
Descripción:	Devuelve *true* si si se puede establecer conexión con el servidor, *false* en caso contrario.
Nombre:	getErrorInfo()
Descripción:	Devuelve la información referente a un posible error ocurrido.

Tabla 20: Descripción de la clase DataQuery.

Nombre: DataQuery		
Tipo de clase	Controladora	
Atributo		**Tipo**
queryClientManager		QueryClientManager
chunkArrived		ChunkArrived
Para cada responsabilidad:		
Nombre:	exec(query: string, block: unsigned int, cursorLength: unsigned int)	
Descripción:	Envía una solicitud de datos al servidor.	
Nombre:	tryRegister()	
Descripción:	Trata de registrar el cliente en el servidor.	

Tabla 21: Descripción de la clase ChunkArrived.

Nombre: ChunkArrived	
Tipo de clase	Controladora
Atributo	**Tipo**
dataReader	DataReader

namesList	vector<string>
typesList	vector<int>
serverError	string
Para cada responsabilidad:	
Nombre:	onChunkArrived(chunk: Chunk)
Descripción:	Recibe el chunk y lo interpreta.
Nombre:	convertTypes(colTypes: vector<string>)
Descripción:	Convierte los identificadores de los tipos de cadenas a numeros enteros.

Tabla 22: Descripción de la clase DataReader.

Nombre: DataReader	
Tipo de clase	Controladora
Atributo	**Tipo**
tupleList	list<Tuple>
drState	unsigned int
drErr	unsigned int
mutex	Mutex
Para cada responsabilidad:	
Nombre:	validateRow(row: int)
Descripción:	Valida el número del registro pasado por parámetros.
Nombre:	validateCol(col: int)
Descripción:	Valida el número de la columna pasado por parámetros.
Nombre:	validateColName(colName: string)
Descripción:	Valida el nombre de la columna pasado por parámetros.
Nombre:	inRow(row: int)

Descripción:	Devuelve el registro correspondiente al número pasado por parámetros.
Nombre:	addTuple(newTuple: Tuple)
Descripción:	Agrega un nuevo registro al DataReader.
Nombre:	erase()
Descripción:	Limpia el DataReader.
Nombre:	row(row: int)
Descripción:	Devuelve el registro correspondiente al número pasado por parámetros.
Nombre:	col(col: int)
Descripción:	Devuelve la columna correspondiente al número pasado por parámetros.
Nombre:	col(colName: string)
Descripción:	Devuelve la columna cuyo nombre es el pasado por parámetros.
Nombre:	field(row: int, col: int)
Descripción:	Devuelve el campo cuya posición dentro del DataReader corresponde a la fila “row” y la columna “col”.
Nombre:	field(row: int, colName: string)
Descripción:	Devuelve el campo cuya posición dentro del DataReader coresponde a la fila “row” y la columna con nombre “colName”.
Nombre:	size()
Descripción:	Devuelve la cantidad de registros.
Nombre:	state()
Descripción:	Devuelve el estado del DataReader.
Nombre:	err()
Descripción:	Devuelve el identificador de un posible error acurrido.

Tabla 23: Descripción de la clase Tuple.

Nombre: Tuple	
Tipo de clase	Controladora
Atributo	**Tipo**
fieldList	list<Field>
tpErr	unsigned int
Para cada responsabilidad:	
Nombre:	validateCol(col: int)
Descripción:	Valida el número de la columna pasado por parámetros.
Nombre:	validateColName(colName: string)
Descripción:	Valida el nombre de la columna pasado por parámetros.
Nombre:	fieldName(col: int)
Descripción:	Devuelve el nombre del campo número “col”.
Nombre:	fieldType(col: int)
Descripción:	Devuelve el tipo del campo número “col”.
Nombre:	fieldType(colName: string)
Descripción:	Devuelve el tipo del campo cuyo nombre es “colName”.
Nombre:	operator [ ](col: int)
Descripción:	Devuelve el campo número “col”.
Nombre:	operator [ ](colName: string)
Descripción:	Devuelve el campo cuyo nombre es “colName”.
Nombre:	value(col: int)
Descripción:	Devuelve el campo número “col”.
Nombre:	value(colName: string)
Descripción:	Devuelve el campo cuyo nombre es “colName”.

Nombre:	size()
Descripción:	Devuelve el número de campos de la tupla.
Nombre:	err()
Descripción:	Devuelve el identificador de un posible error ocurrido.

Tabla 24: Descripción de la clase Field.

Nombre: Field		
Tipo de clase	Controladora	
Atributo		**Tipo**
fieldName		string
fieldType		int
info		char*
Para cada responsabilidad:		
Nombre:	name()	
Descripción:	Devuelve el nombre del campo.	
Nombre:	type()	
Descripción:	Devuelve el tipo del campo.	
Nombre:	c_str()	
Descripción:	Devuelve la información del campo.	
Nombre:	toInt()	
Descripción:	Trata de devolver la información del campo convertida a tipo int, en caso de no ser posible genera un error.	
Nombre:	toFloat()	
Descripción:	Trata de devolver la información del campo convertida a tipo float, en caso de no ser posible genera un error.	

Nombre:	toDouble()
Descripción:	Trata de devolver la información del campo convertida a tipo double, en caso de no ser posible genera un error.
Nombre:	toLongLong()
Descripción:	Trata de devolver la información del campo convertida a tipo long long, en caso de no ser posible genera un error.
Nombre:	toChar()
Descripción:	Trata de devolver la información del campo convertida a tipo char, en caso de no ser posible genera un error.
Nombre:	toString()
Descripción:	Trata de devolver la información del campo convertida a tipo string.
Nombre:	toBool()
Descripción:	Trata de devolver la información del campo convertida a tipo bool, en caso de no ser posible genera un error.

3.4 Diagramas de Secuencia.

Seguidamente se muestran los diagramas de secuencia correspondientes a las principales secciones de los casos de uso.

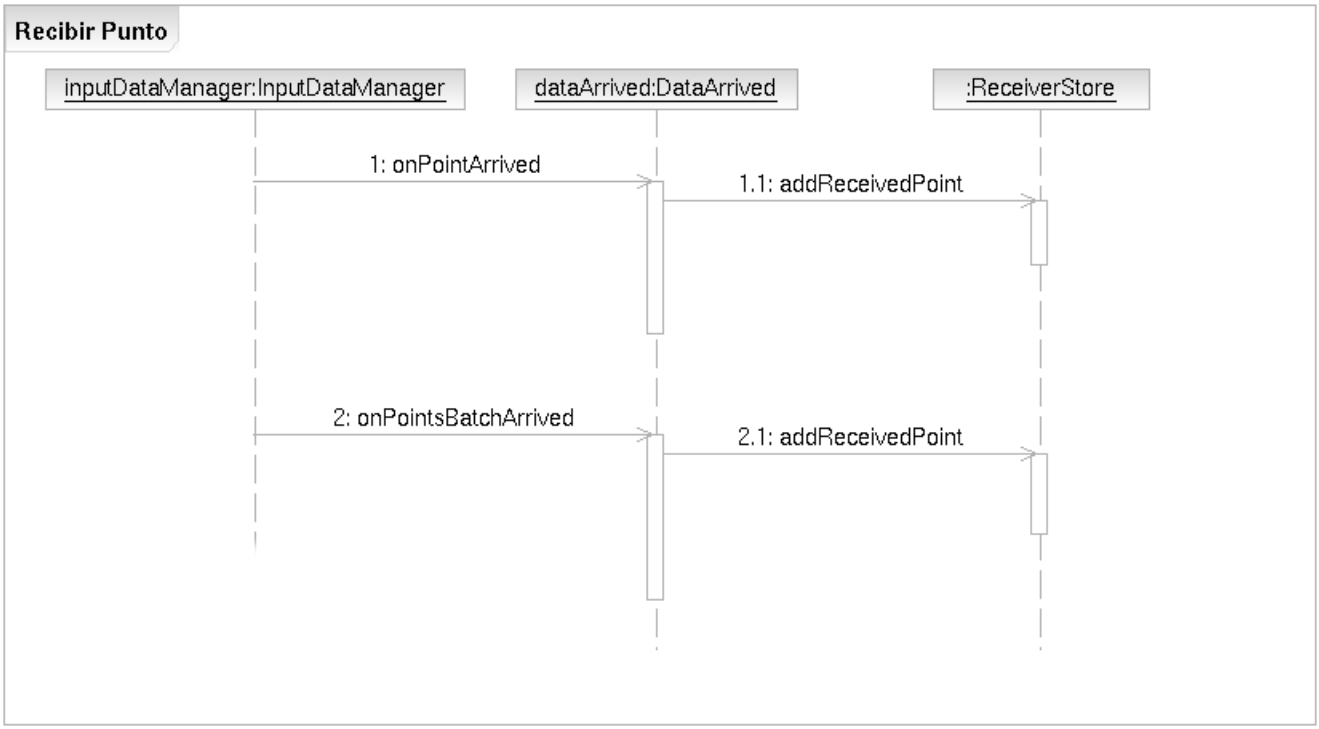

Figura 8: Diagrama de secuencia Recibir Puntos.

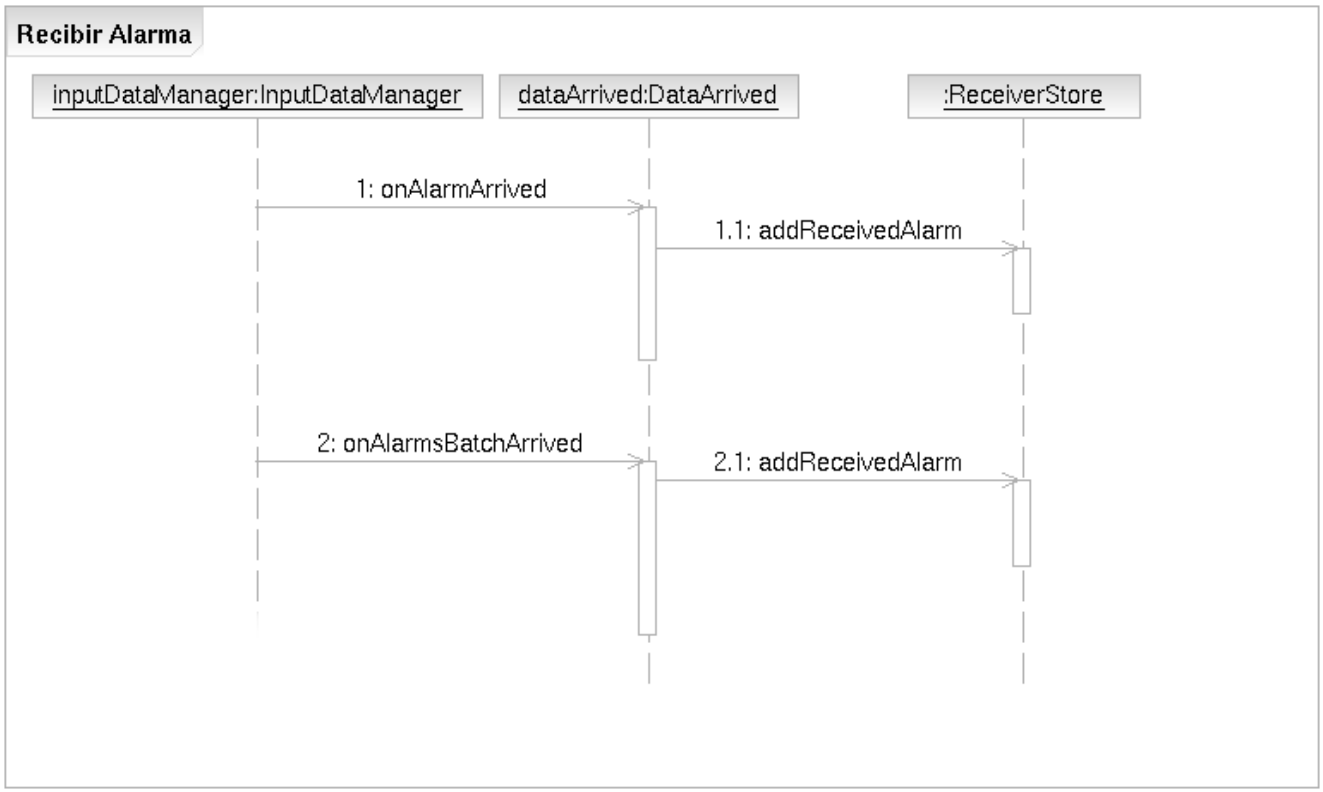

Figura 9: Diagrama de secuencia Recibir Alarmas.

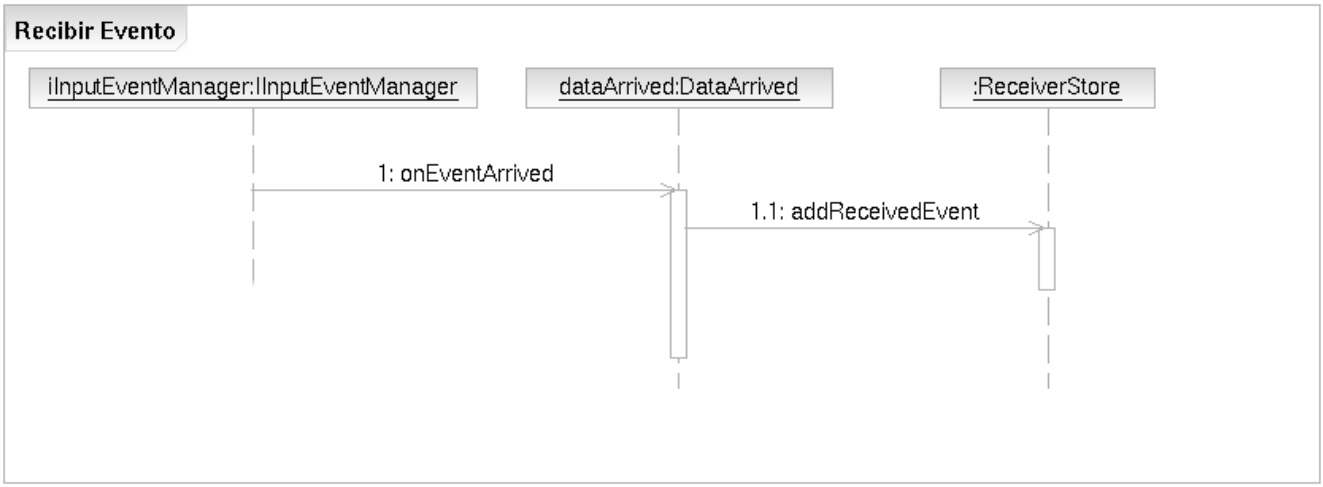

Figura 10: Diagrama de secuencia Recibir Eventos.

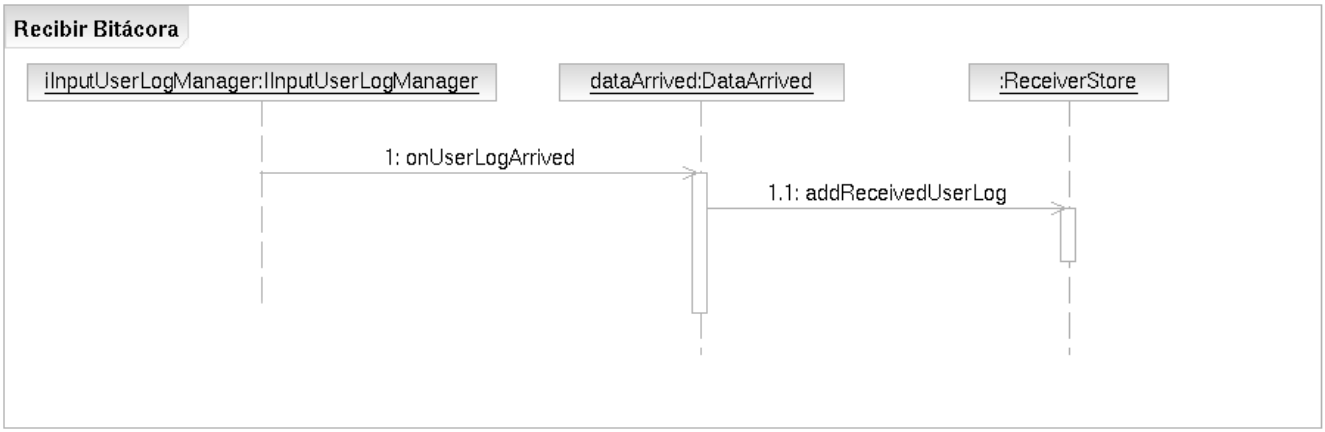

Figura 11: Diagrama de secuencia Recibir Bitácoras.

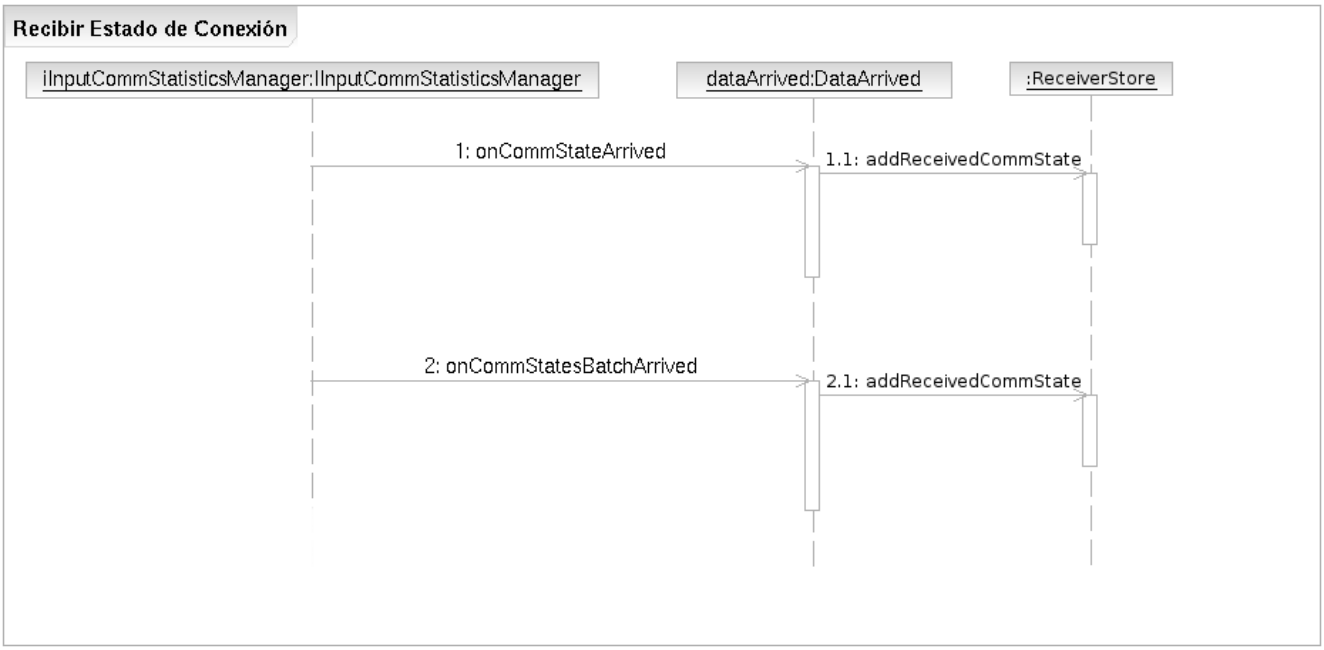

Figura 12: Diagrama de secuencia Recibir Estados de Conexión.

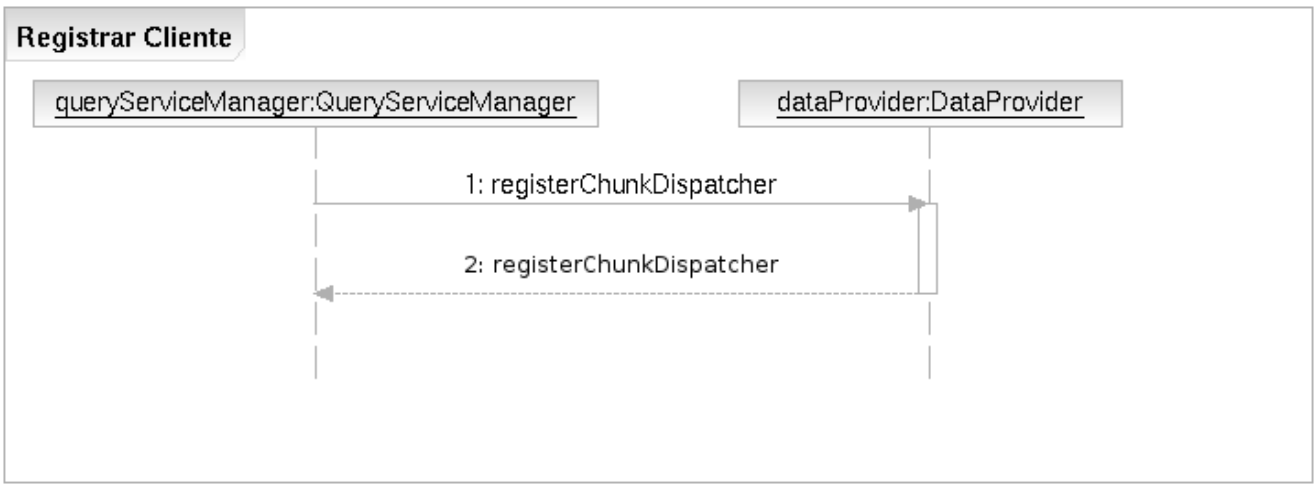

Figura 13: Diagrama de secuencia Registrar Cliente.

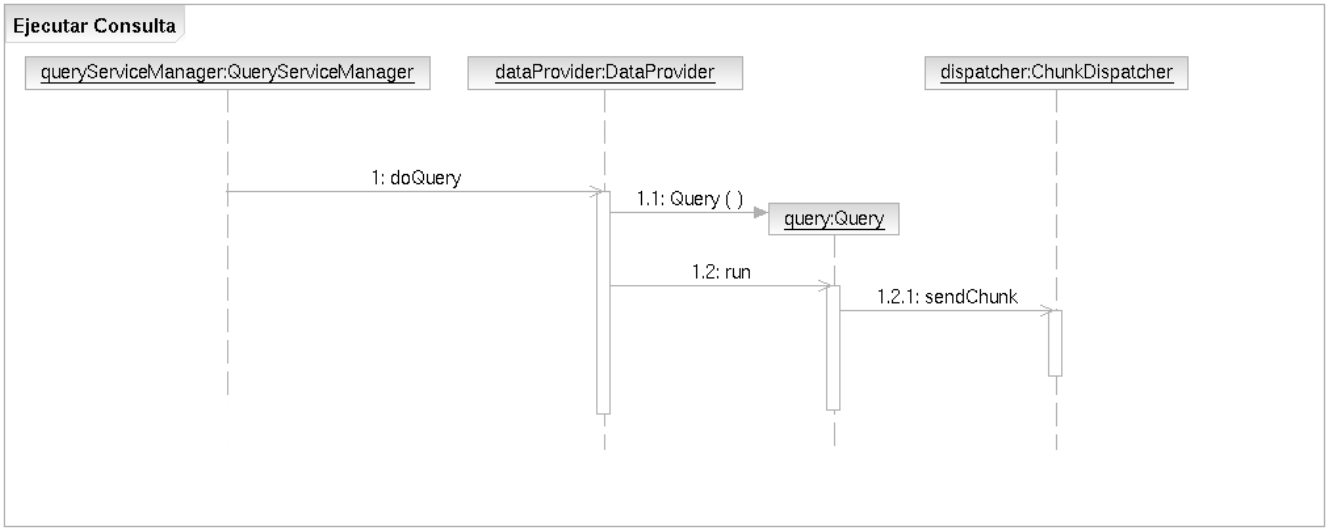

Figura 14: Diagrama de secuencia Ejecutar Consulta.

3.5 Estilo de Codificación. [25]

Nombres.

- Los nombres de las clases son sustantivos singulares.
- Los nombres deben reflejar el “que” y no el “como”.
- 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 nombre 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 de las letras para el caso de los nombres de métodos y funciones. Para el caso de los nombres de variables y atributos debe aplicarse la misma convención, con excepción de la primera letra del nombre, la cual debe ser en minúscula.
- 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 reuso 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, poscondiciones, 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 (4) 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 variables.
- 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.

3.6 Diagramas de componentes.

A continuación se muestran los diagramas de componentes de los paquetes que componen el subsistema, en los mismos se muestra la distribución de las clases en ficheros de código fuente .h y .cpp.

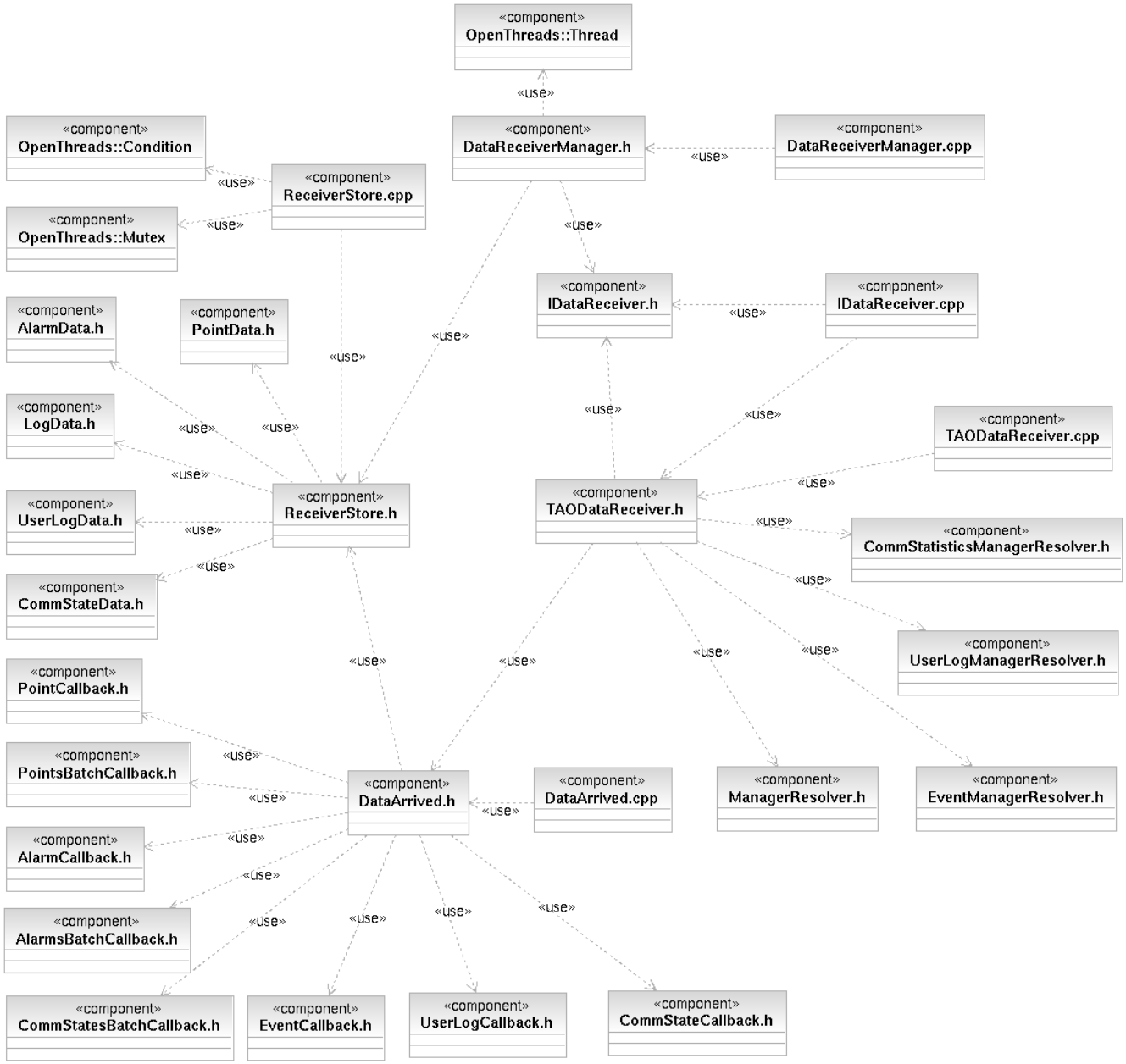

Figura 15: Diagrama de componentes del paquete DataReceiver.

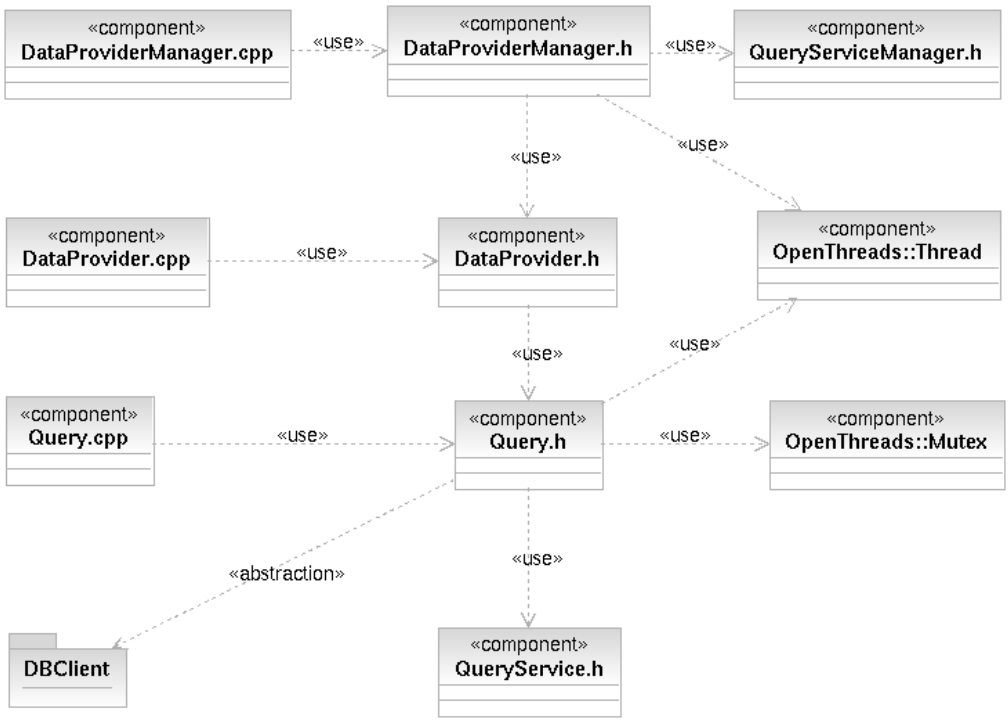

Figura 16: Diagrama de componentes del paquete DataProvider.

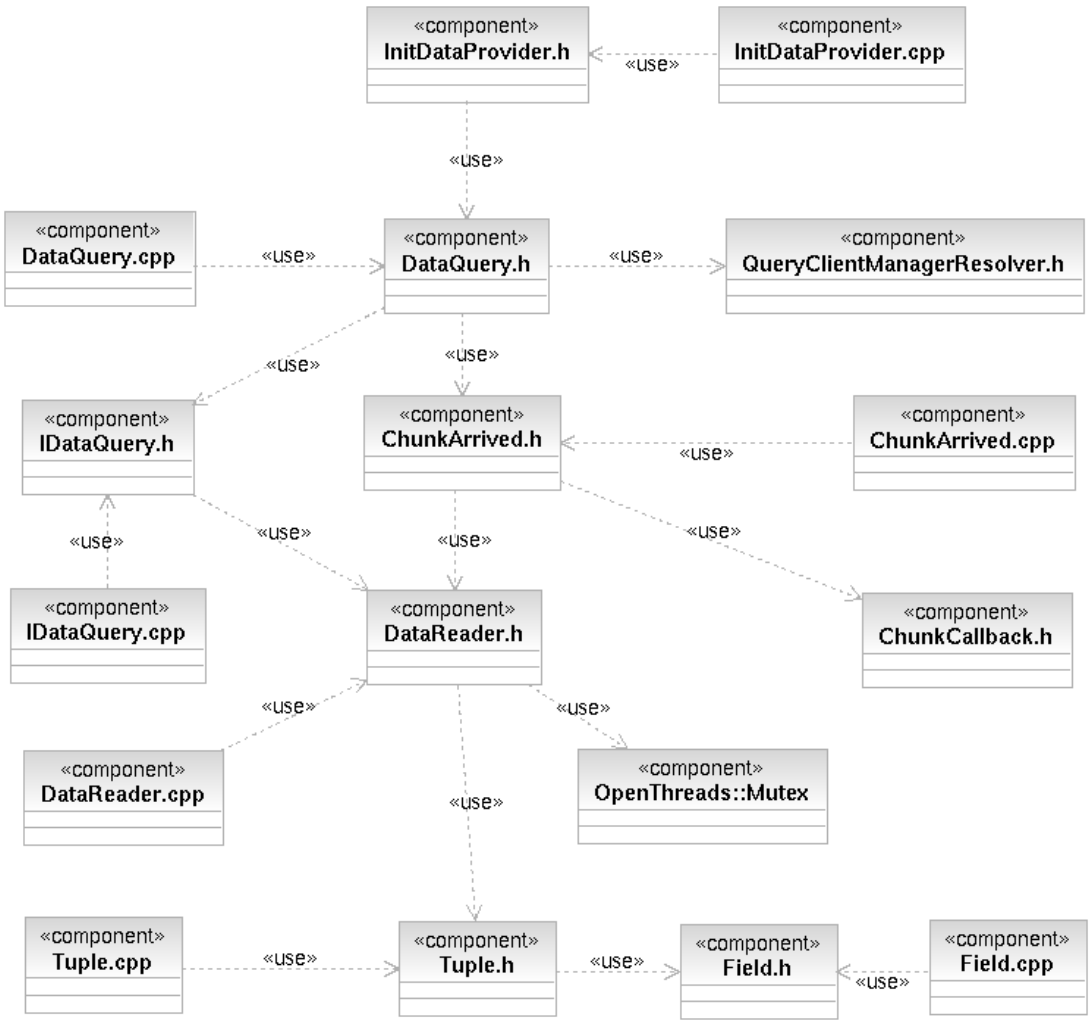

Figura 17: Diagrama de componentes del paquete DataQuery.

Consideraciones finales

Con el desarrollo de este capítulo fueron diseñadas las clases que estructuran los tres paquetes del subsistema de comunicaciones: DataReceiver, DataProvider y DataQuery. Cada uno de ellas con sus respectivas responsabilidades dentro del subsistema:

- DataReceiver: Recepción de datos a través del *Middleware*.
- DataPrivider: Gestión de las consultas a los datos históricos en el servidor.
- DataQuery: Solicitud y gestión de las consultas del lado del módulo cliente.

Finalmente fueron implementadas todas las clases de diseño, teniendo como resultado un subsistema de comunicaciones funcional que cumple con los requisitos funcionales descritos en el capítulo anterior.

CONCLUSIONES

Una vez concluida la presente investigación para el desarrollo de un subsistema de comunicación para el Módulo Base de Datos de Históricos del proyecto “SCADA Nacional”, ha sido cumplido su objetivo general.

En la investigación se arribó a las siguientes conclusiones:

- Se realizó un estudio de la arquitectura propuesta para el SCADA, centrando el mismo en el Módulo BDH. Se describieron los principales servicios que este debe brindar:
  1. Servicios de históricos de variables.
  2. Servicios de históricos de alarmas.
  3. Servicios de históricos de eventos.
  4. Servicios de históricos de bitácoras.
- Se analizaron las interfaces que ofrece *Middleware* para la transmisión de datos, comprendiendo su modo de utilización. Además se identificaron las estructuras en que serian transmitidos estos datos entre los módulos.
- Se diseñaron los tres paquetes que componen el subsistema: DataReceiver, DataProvider y DataQuery, encargados de las tareas de recepción de datos, gestión de consultas del lado del servidor y gestión de consultas del lado del cliente respectivamente.
- Fueron implementados los tres paquetes, obteniendo como resultado final un subsistema de comunicaciones funcional que cumple con los requerimientos especificados.
- Finalmente se integró el subsistema de comunicación al Módulo BDH del SCADA que hoy está siendo probado en varias plantas de Venezuela.

RECOMENDACIONES

El subsistema de comunicación que se obtuvo como resultado de este trabajo, se encuentra actualmente en expansión. En las futuras fases de construcción del proyecto “SCADA Nacional”, se le irán incorporando nuevas funcionalidades.

Se recomiendan los siguientes aspectos para dar continuidad al desarrollo del subsistema:

- Implementar la recepción de datos de múltiples módulos BDTR.
- En el paquete DataQuery, permitir la gestión de varias consultas de modo concurrente.
- Optimizar el manejo de errores en las solicitudes de datos históricos.

REFERENCIAS BIBLIOGRÁFICAS

1. **Gustabello, Robby, y otros.** *Desarrollo de Módulo de Base de Datos Históricos para Sistema SCADA.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2007.

2. Introducción a los sistemas de control supervisor y de adquisición de datos (SCADA). *InfoPLC.net.* [En línea] 2004. [Citado el: 12 de Marzo de 2008.] http://www.infoplc.net/Documentacion/Docu_SCADA/infpPLC_net_Introduccion_Sistemas_SCADA.html.

3. Scadas. *automatas.org.* [En línea] 2006. [Citado el: 23 de Marzo de 2008.] http://www.automatas.org/redes/scadas.htm.

4. **Romagosa, Jaume, Gallego, David y Pacheco, Rául.** *Sistemas SCADA.* s.l. : Universidad Politécnica de Cataluña, 2004.

5. SCADA Systems, Data Acquisition Systems, Telemetry Hardware, Choosing Scada . *epgco.com.* [En línea] EPG Companies Inc. [Citado el: 28 de Marzo de 2008.] http://www.epgco.com/scada-system-assessment.html.

6. **Herrera, Moisés.** *Especificación de Históricos del Sistema SCADA v1.2.* 2008.

7. *Historical Data Access from Industrial Systems Specification, HDAIS.* s.l. : OMG, 2005.

8. **Charrouf, Raed.** *Especificación para Sumarios de Alarmas v1.5.* 2008.

9. **Barrientos, Mildred.** *Especificación de Eventos v1.5.* 2008.

10. *OPC Historical Data Access.* s.l. : OPC Foundation, 2003.

11. *dst.pdvsa.com.* [En línea] [Citado el: 2 de Abril de 2008.] http://trac.dst.pdvsa.com/scadanac/wiki/Arquitectura/VistaDespliegue.

12. Data Provider. *dst.pdvsa.com.* [En línea] [Citado el: 2 de Abril de 2008.] http://trac.dst.pdvsa.com/scadanac/wiki/AnalisisDiseno/BaseDeDatosHistorico/DataProvider.

13. Data Receiver. *dst.pdvsa.com.* [En línea] [Citado el: 2 de Abril de 2008.] http://trac.dst.pdvsa.com/scadanac/wiki/AnalisisDiseno/BaseDeDatosHistorico/DataReceiver.

14. Object Repository. *dst.pdvsa.com.* [En línea] [Citado el: 3 de Abril de 2008.] http://trac.dst.pdvsa.com/scadanac/wiki/AnalisisDiseno/BaseDeDatosHistorico/ObjectRepository.

15. Garbage Collector. *dst.pdvsa.com.* [En línea] [Citado el: 3 de Abril de 2008.] http://trac.dst.pdvsa.com/scadanac/wiki/AnalisisDiseno/BaseDeDatosHistorico/GarbageCollector.

16. DBClient. *dst.pdvsa.com.* [En línea] [Citado el: 3 de Abril de 2008.] http://trac.dst.pdvsa.com/scadanac/wiki/AnalisisDiseno/BaseDeDatosHistorico/DBClient.

17. Alta Disponibilidad . *dst.pdvsa.com.* [En línea] [Citado el: 3 de Abril de 2008.] http://trac.dst.pdvsa.com/scadanac/wiki/AnalisisDiseno/AltaDisponibilidad .

18. Multi-Threaded Programming With POSIX Threads. *actcom.co.il.* [En línea] [Citado el: 15 de Abril de 2008.] http://users.actcom.co.il/~choo/lupg/tutorials/multi-thread/multi-thread.html#definition.

19. Recibir Puntos Alarmas. *dst.pdvsa.com.* [En línea] [Citado el: 17 de Abril de 2008.] http://trac.dst.pdvsa.com/scadanac/wiki/AnalisisDiseno/Middleware/RecibirPuntosAlarmas.

20. Recibir Eventos. *dst.pdvsa.com.* [En línea] [Citado el: 17 de Abril de 2008.] http://trac.dst.pdvsa.com/scadanac/wiki/AnalisisDiseno/Middleware/RecibirEventos.

21. Recibir Bitacoras. *dst.pdvsa.com.* [En línea] [Citado el: 17 de Abril de 2008.] http://trac.dst.pdvsa.com/scadanac/wiki/AnalisisDiseno/Middleware/RecibirBitacoras.

22. Middleware. *dts.pdvsa.com.* [En línea] [Citado el: 20 de Abril de 2008.] http://trac.dst.pdvsa.com/scadanac/wiki/AnalisisDiseno/Middleware.

23. OpenThreads - A cross-platform, lightweigth C++ thread API. *sourceforge.net.* [En línea] [Citado el: 11 de Abril de 2008.] http://openthreads.sourceforge.net/.

24. **Villarreal, Elizabeth.** *Especificación de Requerimientos de Software v6.9.* s.l. : DST-AIT, 2008.

25. *Guía de Estilo de Código.* s.l. : DST-AIT, 2007.

BIBLIOGRAFÍA CONSULTADA

1. **King, Peter.** *SCADA Systems – Looking Ahead.* s.l. : Control Microsystems Inc., 2005.

2. **Barr, Dale.** *Supervisory Control and Data Acquisition (SCADA) Systems.* s.l. : Communication Technologies Inc., 2004.

3. *Introducción a la Seguridad en Sistemas SCADA.* s.l. : NeutralBit, 28 de 06 de 2006.

4. *Software de adquisición, supervisión y control.* **Ayza, Jordi.** 299, s.l. : Tecnipublicaciones España, Septiembre de 1999, Automática e Instrumentación. 0213-3113.

5. **Park, John y Mackay, Steve.** *Practical Data Acquisition for Instrumentation and Control Systems.* s.l. : Elsevier, 2003. 07506 57960.

6. **Macchi, José y Mezzanotte, Martín.** *FUNDAMENTOS BÁSICOS DE THREADS.* Buenos Aires : Universidad Nacional del Centro de la Provincia de Buenos Aires, 2006.

7. **Pérez, Maikel.** *Informe de Pruebas del Middleware v1.1.* 2008.

8. Multiprograma & Multitarea. *zator.com.* [En línea] Zator Systems. [Citado el: 11 de Abril de 2008.] http://www.zator.com/Cpp/E1_7b.htm.

9. SCADA/EMS History. *scadahistory.com.* [En línea] [Citado el: 27 de Marzo de 2008.] http://scadahistory.com/resources/SCADA.htm.

GLOSARIO DE TÉRMINOS

1. **Autenticaciones**: Certificaciones de que los usuarios son efectivamente quienes dicen ser. Por ejemplo: Un usuario introduce su contraseña y el sistema se encarga de verificar que sea la correcta. Si esto se cumple, el sistema puede darle acceso a un determinado recurso.
2. **Autómatas Programables**: Puede definirse como un equipo electrónico programable, por lo general en lenguaje no informático y diseñado para controlar, en tiempo real y en ambiente industrial, procesos secuenciales.
3. **Buffers**: En informática, es un espacio de memoria, en el que se almacenan datos.
4. **Hot Standby**: Espera activa. En la redundancia de sistemas, un sistema auxiliar en espera que puede entrar a funcionar inmediatamente en caso de que ocurra una falla en el sistema principal.
5. **LAN**: Son las siglas de *Local Area Network*, Red de área local. Una LAN es una red que conecta los ordenadores en un área relativamente pequeña y predeterminada (como una habitación, un edificio, o un conjunto de edificios).
6. **Multiplataforma**: Término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas. Por ejemplo, una aplicación multiplataforma podría ejecutarse en Windows en un procesador x86, en GNU/Linux en un procesador x86, y en Mac OS X en uno x86.
7. **Sección Crítica**: Porción de código de un programa de computador el cual accede a un recurso compartido (estructura de datos o dispositivo) que no debe de ser accedido por más de un hilo en ejecución.
8. **Replicación**: Consiste en el transporte de datos entre dos o más servidores, permitiendo que ciertos datos de la base de datos estén almacenados en más de un sitio, y así aumentar la disponibilidad de los datos y mejorar el rendimiento de las consultas globales
9. **SQL**: Son las siglas de Structured Query Language, Lenguaje de consulta estructurado, es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar -de una forma sencilla- información de interés de una base de datos, así como también hacer cambios sobre la misma.
10. **Sumarizadas**: Concepto manejado dentro del Módulo BDH. Es utilizado para nombrar al proceso de compresión, de las variables que se almacenan en un sistema de tablas jerárquicas dentro de la BD.
11. **Telemetría**: Tecnología que nos permite la medición remota de magnitudes físicas y el posterior envío de la información hacia el operador de determinado sistema. La palabra *telemetría* procede de las palabras griegas *tele* ("lejos") y *metron* ("medida").

12. **WAN**: Son las siglas de *Wide Area Network*, Red de área amplia, una red de ordenadores que abarca un área geográfica relativamente grande. Normalmente, un WAN consiste en dos o más redes de área local (LANs).
13. **XML**: Son las siglas de *Extensible Markup Language*, Lenguaje de marcas extensible, es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium*. Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos.
14. **XQuery**: Lenguaje de Consulta XML (XML Query). Es un lenguaje que ofrece la posibilidad de realizar consultas en infinidad de tipos diferentes de documentos como son documentos estructurados, colecciones de documentos, bases de datos, catálogos, etc., para extraer datos en la Web.