# Universidad de las Ciencias Informáticas

## Facultad 5

**TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS.**

**Título:** Módulo de gestión y archivo de datos para el SCADA Ligero.


**Autor:**

Luis Enrique Nerey Cruz.

**Tutor:**

Ing. Evián Suárez Rodríguez.


**Ciudad de La Habana**

**2012**

## DECLARACIÓN DE AUTORÍA.

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

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

Luis Enrique Nerey Cruz — Ing. Evián Suárez Rodríguez

_____ _____

Firma del Autor — Firma del Tutor

## DATOS DE CONTACTO.

**Nombre y apellidos:** Evián Suárez Rodríguez.

**Institución:** Universidad de las Ciencias Informáticas (UCI).

**Título:** Ingeniero en Ciencias Informáticas.

**e-mail:** esrodriguez@uci.cu

**Cargo:** Líder de la línea Comunicaciones, del DPTO Construcciones de Componentes, del Centro de Desarrollo de Informática Industrial.

**AGRADECIMIENTOS.**

*A mi mamá por ser mi guía durante toda mi vida.*

*A mi papá y Yamila por todos los consejos que me dieron durante tanto tiempo que me ayudaron a ser alguien mejor.*

*A mi hermanita por tanto cariño. A mi abuela por ser tan preocupada.*

*A Yuri por estar a mi lado siempre durante todo este año, tanto en Venezuela como aquí en la escuela. Por todo el cariño que me ha dado siempre y todas las cosas que me ha enseñado. Gracias por ser esa conciencia que siempre me decía que tenía que trabajar en la tesis para terminarla.*

*A mi abuelo Mayo que fue una guía a seguir durante todo este tiempo.*

*A mis abuelos que aunque están lejos siempre están preocupados por mí.*

*A mi primo por ser el otro hermano que siempre tuve.*

*A Raúl que aunque no somos hermanos de sangre siempre lo seremos de corazón.*

*A mis amigos de la escuela que ayudan siempre a pasar la universidad mucho mejor.*

*Gracias a todos que me han apoyado durante mi vida entera. Muchas gracias.*

## DEDICATORIA.

*A mi familia completa*

# RESUMEN.

El presente trabajo de diploma se centra en el desarrollo de un módulo de gestión y archivo de datos para el sistema de control, supervisión y de adquisición de datos (SCADA) Ligero, del Centro de Informática Industrial (CEDIN). El módulo permite la gestión de datos históricos, a través del uso de un sistema de gestión de base de datos embebido. En el mismo se pueden encontrar aspectos generales sobre los sistemas SCADA, de algunos sistemas de gestión de base de datos (SGBD) y algunas bibliotecas gráficas de desarrollo.

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

**Palabras claves:**

Base de datos histórico, embebido, SCADA, sistemas de gestión de base de datos.

# ÍNDICE

**DECLARACIÓN DE AUTORÍA.** ..... **II**

**DATOS DE CONTACTO.** ..... **III**

**AGRADECIMIENTOS.** ..... **IV**

**DEDICATORIA.** ..... **V**

**RESUMEN.** ..... **VI**

**ÍNDICE** ..... **VII**

**ÍNDICE DE FIGURAS** ..... **XI**

**ÍNDICE DE TABLAS** ..... **XII**

**INTRODUCCIÓN** ..... **1**

**CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.** ..... **4**

1.1 INTRODUCCIÓN. ..... 4

1.2 SISTEMAS SCADA. ..... 4

*1.2.1 Descripción.* ..... *4*

*1.2.2 Funcionalidades.* ..... *5*

*1.2.3 Estructura.* ..... *6*

1.3 MÓDULO DE BASE DE DATOS HISTÓRICOS. ..... 7

*1.3.1 Servicio histórico de variables.* ..... *7*

*1.3.2 Servicio histórico de alarmas.* ..... *10*

*1.3.3 Servicio histórico de eventos.* ..... *10*

*1.3.4 Servicio histórico de bitácoras.* ..... *11*

*1.3.5 Solicitud de información de los históricos de variables.* ..... *12*

1.4 SISTEMAS DE GESTIÓN DE BASE DE DATOS. ..... 13

*1.4.1 Base de datos PostgreeSQL.* ..... *13*

1.4.1.1 Descripción. ..... 13

1.4.1.2 Características. ..... 13

1.4.1.3 Limitaciones. ..... 14
*1.4.2 Base de datos SQLite. ..... 14*
1.4.2.1 Descripción. ..... 15
1.4.2.2 Características. ..... 15
1.4.2.3 Limitaciones. ..... 16
*1.4.3 Base de datos Derby. ..... 16*
1.4.3.1 Descripción. ..... 16
1.4.3.2 Características. ..... 17
1.4.3.3 Limitaciones. ..... 18
*1.4.4 Base de datos Berkeley. ..... 18*
1.4.4.1 Descripción. ..... 18
1.4.4.2 Características. ..... 18
1.4.4.3 Limitaciones. ..... 19
*1.4.5 Selección del sistema gestor de base de datos. ..... 19*
1.5 BIBLIOTECAS GRÁFICAS DE DESARROLLO. ..... 20
*1.5.1 Qt. ..... 20*
*1.5.2 GTK+. ..... 20*
1.6 TECNOLOGÍAS Y HERRAMIENTAS A UTILIZAR PARA EL DESARROLLO. ..... 21
*1.6.1 Microsoft Visual C++ 2008 como IDE de desarrollo. ..... 21*
*1.6.2 Qt. como biblioteca gráfica. ..... 21*
*1.6.3 C++ como lenguaje de programación. ..... 22*
*1.6.4 Visual Paradigm como herramienta CASE. ..... 22*
*1.6.5 UML como lenguaje de modelado. ..... 23*
*1.6.6 RUP como metodología de desarrollo. ..... 23*
*1.6.7 Berkeley como sistema de base de datos embebido. ..... 24*
1.7 CONSIDERACIONES PARCIALES DEL CAPÍTULO. ..... 24
**CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA. ..... 25**
2.1 INTRODUCCIÓN. ..... 25
2.2 SOLUCIONES TÉCNICAS. ..... 25
2.3 MODELO DEL DOMINIO. ..... 25

*2.3.1 Clases conceptuales o glosario de términos del dominio. ..... 26*
2.4 DESCRIPCIÓN DEL SISTEMA PROPUESTO. ..... 27
*2.4.1 Requisitos funcionales. ..... 27*
*2.4.2 Requisitos no funcionales. ..... 28*
*2.4.3 Actores del sistema. ..... 29*
*2.4.4 Casos de uso del sistema. ..... 30*
*2.4.5 Diagramas de los casos de uso del sistema. ..... 30*
*2.4.6 Descripción de los casos de uso del sistema. ..... 31*
2.4.6.1 RF Almacenar variable. ..... 31
2.4.6.2 RF Definir recolector de basura. ..... 33
2.4.6.3 RF Devolver variables almacenadas. ..... 34
2.4.6.4 RF Mostrar variables en un intervalo de tiempo. ..... 35
2.4.6.5 RF Mostrar tendencia de una variable en un intervalo de tiempo. ..... 36
2.4.6.6 RF Mostrar agregados. ..... 38
2.5 CONSIDERACIONES PARCIALES DEL CAPÍTULO. ..... 41
**CAPÍTULO 3: DISEÑO, IMPLEMENTACIÓN Y PRUEBAS ..... 42**
3.1 INTRODUCCIÓN. ..... 42
3.2 DISEÑO DEL SISTEMA. ..... 42
*3.2.1 Paquete View. ..... 44*
*3.2.2 Paquete DBClient. ..... 46*
*3.2.3 Subpaquete BerkeleyDB. ..... 47*
*3.2.4 Paquete DataRecords. ..... 48*
3.3 IMPLEMENTACIÓN DEL SISTEMA. ..... 48
*3.3.1 Estilo de código. ..... 48*
*3.3.2 Diagrama de componentes. ..... 51*
3.4 REALIZACIÓN DE PRUEBAS. ..... 52
3.5 CONSIDERACIONES PARCIALES. ..... 56
**CONCLUSIONES ..... 57**
**RECOMENDACIONES ..... 58**

**ÍNDICE**

**REFERENCIAS BIBLIOGRÁFICAS. ..... 59**

**BILIOGRAFÍA. ..... 60**

**GLOSARIO DE TÉRMINOS ..... 61**

# ÍNDICE DE FIGURAS

Figura 1: Sistema SCADA. ..... 5
Figura 2: Esquema de comunicación entre los diferentes módulos que intervienen en el proceso de recolección y archivo de históricos. ..... 9
Figura 3: Modelo de dominio. ..... 26
Figura 4: Diagrama de caso de uso del sistema (operador). ..... 31
Figura 5: Diagrama de caso de uso del sistema (sistema) ..... 31
Figura 6: Arquitectura en capas del sistema. ..... 42
Figura 7: Diagrama de clases del paquete View. ..... 44
Figura 8: Diagrama de clases del paquete DBClient. ..... 46
Figura 9: Diagrama de clases del subpaquete BerkeleyDB. ..... 47
Figura 10: Diagrama de clases del subpaquete DataRecords. ..... 48
Figura 11: Diagrama de componentes del sistema. ..... 52

# ÍNDICE DE TABLAS

Tabla 1: Limitaciones de PostgreeSQL. 14
Tabla 2: Características de los esquemas. 18
Tabla 3: Descripción de los actores. 30
Tabla 4: Clasificación de los casos de uso del sistema. 30
Tabla 5: Descripción del caso de uso “Almacenar variable”. 33
Tabla 6: Descripción del caso de uso “Definir recolector de basura”. 34
Tabla 7: Descripción del caso de uso “Devolver variables almacenadas”. 35
Tabla 8: Descripción del caso de uso “Mostrar variables en un intervalo de tiempo”. 36
Tabla 9: Descripción del caso de uso “Mostrar tendencia de una variable en un intervalo de tiempo”. 38
Tabla 10: Descripción del caso de uso “Mostrar agregados”. 41
Tabla 11: Descripción de las clases del paquete View. 45
Tabla 12: Descripción de las clases del paquete DBClient. 47
Tabla 13: Descripción de las clases del paquete DataRecords. 48
Tabla 14: Pruebas realizadas al sistema. 55
Tabla 15: Pruebas realizadas a la aplicación. 56

# INTRODUCCIÓN

La Universidad de las Ciencias Informáticas (UCI), tiene entre sus principales objetivos producir software y servicios informáticos a través de su modelo de formación que incluye la vinculación estudio – trabajo [1]. Dentro de este modelo la Facultad 5 integra el CEDIN, el cual tiene entre sus proyectos el SCADA "Guardián del ALBA", que fue diseñado e implementado con el apoyo de la compañía petrolera estatal venezolana (PDVSA) para monitorear la extracción de petróleo.

Los sistemas SCADA son una aplicación de software especialmente diseñada para funcionar sobre ordenadores en el control de producción, proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, entre otros.) y controlando el proceso de forma automática desde una computadora [2] [3].

Los sistemas de supervisión y control necesitan almacenar los datos y cambios de estado dentro del sistema, posibilitando el análisis de la información histórica que se tiene del proceso para la toma de decisiones y la generación de reportes a distintos niveles.

Con la experiencia adquirida en el centro con el desarrollo del SCADA, se incrementan las posibilidades de negocios, creando la necesidad de desplegarlo en varias entidades del país.

En el CEDIN se ha desarrollado recientemente un SCADA Ligero para controlar procesos en los que no existen una gran cantidad de variables, con el objetivo de emplearlo en los escenarios que por falta de recursos necesitan una solución más sencilla. Dicho SCADA Ligero no cuenta con un mecanismo que almacene toda la información que maneja y ante la existencia de un error o problema en el sistema, al operador se le hace imposible analizar todo lo que ocurrió mediante las trazas por lo que se dificulta la toma correcta de decisiones.

El SCADA "Guardián del ALBA" tiene entre sus módulos el de base de datos histórico (BDH) encargado del almacenamiento de los datos y cambios de estado dentro del sistema, dicho módulo cuenta con muchas funcionalidades y agregados que no son necesarios utilizar ya que el SCADA Ligero va a ser utilizado fundamentalmente en procesos que presenten pocas variables. Además de sus considerables

requerimientos de memoria y procesamiento hace que constituyan una solución poco eficiente en esquemas simples de almacenamiento. Presentes fundamentalmente en empresas donde la disponibilidad de recursos de software y hardware es muy limitada y sólo requieren el manejo de pocos datos con bajos tiempos de acceso.

A raíz de lo anteriormente planteado se define como **problema a resolver:** ¿Cómo gestionar y archivar la información obtenida por el SCADA Ligero para su posterior análisis histórico?

Del problema anterior se define que el **objeto de estudio** es: La gestión de datos.

Para dar solución al problema planteado se propone como **objetivo general** desarrollar un módulo de gestión y archivo de datos que permita el almacenamiento de forma eficiente y que pueda gestionar la información a partir de una interfaz visual independiente al sistema.

Tomando como **campo de acción:** La gestión y archivo de datos en sistemas SCADA.

Para dar cumplimiento al objetivo general se plantean las siguientes **tareas de investigación**:

- ✓ Elaboración del marco teórico de la investigación para formular la base del desarrollo del sistema.
- ✓ Selección de las funcionalidades básicas que necesita un módulo de gestión y archivo de datos para su posterior desarrollo.
- ✓ Identificación del sistema de gestión de base de datos más factible para su uso en el módulo.
- ✓ Selección de las herramientas libres necesarias y apropiadas para el desarrollo.
- ✓ Descripción del sistema propuesto para un mejor entendimiento.
- ✓ Elaboración de una biblioteca para el almacenamiento de datos.
- ✓ Elaboración de una aplicación para visualizar la información almacenada.
- ✓ Realización de pruebas al módulo de gestión y archivo de datos para validar su correcto desempeño.

- ✓ Realización de pruebas a la aplicación visual para validar su correcto funcionamiento.

Entre los **métodos científicos** utilizados se destacan los siguientes:

**Teóricos:**

- ✓ **Histórico-Lógico:** Empleado durante la investigación de los antecedentes y tendencias actuales de los módulos de base de datos históricos. Además en la profundización de los conceptos, términos y vocabulario del propio objeto de estudio y el campo de acción.
- ✓ **Analítico-Sintético:** Durante la definición de los requerimientos y funcionalidades básicas del módulo de gestión y archivo de datos para el SCADA Ligero.

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

**En el capítulo 1:** Fundamentación teórica, se describen las principales características de un SCADA, así como los sistemas gestores de base de datos utilizados en la investigación. Se describen además las técnicas y herramientas utilizadas.

**En el capítulo 2:** Características del sistema, se ofrece una visión práctica del sistema. Se definen los requisitos funcionales y no funcionales. Finalmente, se determinan los casos de uso y se describe cada uno.

**En el capítulo 3:** Diseño, implementación y pruebas, se diseña un sistema de clases, en correspondencia con las técnicas de la programación orientada a objetos, que luego son implementadas teniendo como resultado final un prototipo funcional del módulo. Además se realizan las pruebas de funcionalidad para validar la implementación.

Por último, como parte de los anexos, se muestra un glosario de términos y abreviaturas que facilita la comprensión del lenguaje técnico y las abreviaturas utilizadas en el trabajo.

# CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

## 1.1 Introducción.

En este capítulo se abordan conceptos esenciales para la comprensión del problema a resolver. Se abordan aspectos generales de los SCADA, de los módulos de base de datos históricos y se exponen las características de algunos SGBD empotrados. También se muestran aspectos relacionados con las técnicas y herramientas utilizadas en el desarrollo.

## 1.2 Sistemas SCADA.

SCADA es el acrónimo de "*Supervisory Control And Data Acquisition*" (control, supervisión y adquisición de datos). Los sistemas SCADA utilizan la computadora y tecnologías de comunicación para automatizar el monitoreo y control de procesos industriales. Estos sistemas son partes importantes de la mayoría de los ambientes industriales complejos o muy geográficamente dispersos, ya que pueden recolectar la información de una gran cantidad de dispositivos rápidamente y presentarla a un operador en una forma amigable [4].

### 1.2.1 Descripción.

Los sistemas SCADA son una aplicación o un conjunto de ellas diseñadas especialmente para funcionar sobre ordenadores en el control de producción, proveyendo comunicación con los dispositivos de campo (controladores, autómatas, sensores, actuadores, registradores, entre otros) y controlando el proceso de forma automática desde la pantalla del ordenador.

En este tipo de sistemas usualmente existe un ordenador que efectúa tareas de supervisión y gestión de alarmas, así como tratamiento de datos y control de procesos. La comunicación entre los dispositivos de campo y la terminal central de procesamiento se realiza mediante buses especiales o a través de redes LAN. Todo esto se ejecuta normalmente en tiempo real y están diseñados para dar al operador de planta la posibilidad de supervisar y controlar dichos procesos [5].

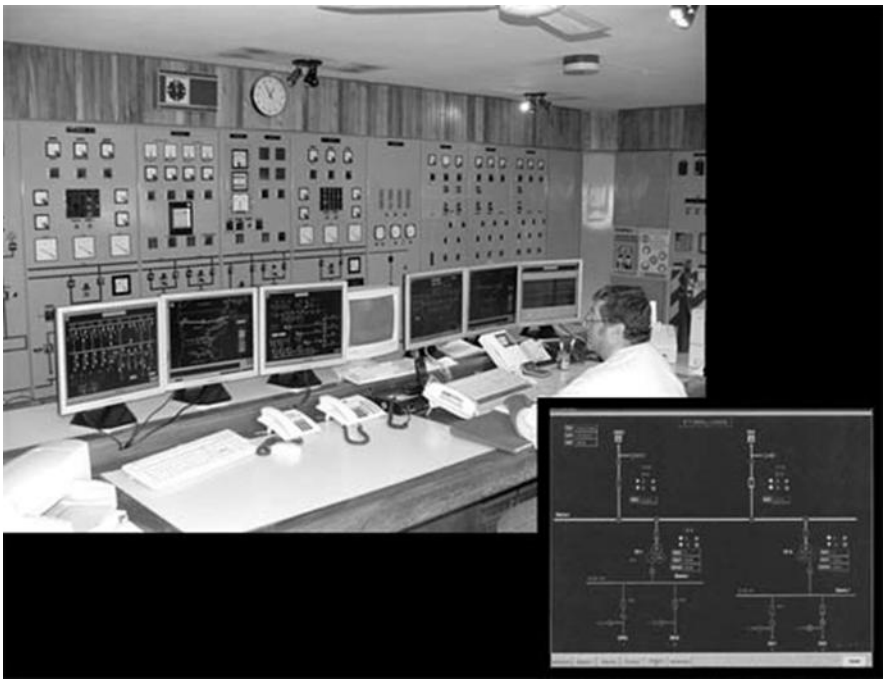


**Figura 1: Sistema SCADA.**

### 1.2.2 Funcionalidades.

El SCADA como sistema, comprende una serie funcionalidades encaminadas a establecer una comunicación lo más clara posible entre el proceso y el operador. Algunas de las principales prestaciones son las siguientes [5]:

- ✓ **Adquisición de datos:** incluye recolectar, procesar, almacenar y mostrar la información recibida en forma continua desde los equipos de campo.
- ✓ **Supervisión:** el operador podrá observar desde el monitor la evolución de las variables de control, como cambios que se produzcan en la operación diaria de la planta.
- ✓ **Control:** el operador puede ejecutar acciones de control que podrán modificar la evolución del proceso en situaciones irregulares que se generen.
- ✓ **Generación de reportes:** de los datos adquiridos se pueden generar representaciones gráficas de los datos, predicciones, control estadístico, gestión de la producción, gestión administrativa y financiera.

### 1.2.3 Estructura.

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

Entre los elementos de hardware pueden destacarse [5]:

- ✓ **Unidad terminal maestra (MTU):** es el computador principal del sistema el cual supervisa y recoge la información del resto de las subestaciones; soporta una interfaz hombre máquina.
- ✓ **Unidad remota de telemetría (RTU):** es un dispositivo instalado en una localidad remota del sistema y está encargado de recopilar datos para luego ser transmitidos hacia la Unidad Terminal Maestra.
- ✓ **Red de comunicación:** el sistema de comunicación es el encargado de la transferencia de información entre la planta y la arquitectura hardware que soporta el SCADA.
- ✓ **Instrumentación de campo:** están constituidos por todos aquellos dispositivos que permiten tanto realizar la automatización o control del sistema.

Los componentes de software también conocidos como módulos, varían un tanto en dependencia del sistema del que se trate. A pesar de ello en casi todos pueden encontrarse los siguientes [6]:

- ✓ **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 sinópticos gráficos almacenados en el ordenador de proceso y generados desde el editor incorporado en el SCADA o importados desde otra aplicación durante la configuración del paquete.
- ✓ **Módulo de proceso:** ejecuta las acciones de mando pre-programadas a partir de los valores actuales de variables leídas.
- ✓ **Gestión y archivo de datos:** se encarga del almacenamiento y procesado ordenado de los datos, de forma que otra aplicación o dispositivo pueda tener acceso a ellos.
- ✓ **Comunicaciones:** se encarga de la transferencia de información entre la planta y la arquitectura hardware que soporta el SCADA, y entre ésta y el resto de elementos informáticos de gestión.

## 1.3 Módulo de base de datos históricos.

En los sistemas SCADA la recepción de los datos desde los niveles de campo hasta los niveles gerenciales se perfecciona como el servicio más utilizado en los últimos tiempos, sobresaliendo la captura y visualización en tiempo real en las consolas de operación. Sin embargo en los sistemas donde se necesite un análisis de la información histórica recolectada por los dispositivos de campo, debe existir un mecanismo que permita almacenar esos datos.

La información almacenada es utilizada por una serie de aplicaciones entre las cuales se destacan los servicios gerenciales, como la gestión de producción, mantenimiento y control. Utilizando algoritmos inteligentes, predictivos y adaptativos. La utilidad más inmediata es la generación de reportes por parte de los operadores y usuarios del SCADA [7]. Existen en la actualidad varios tipos de históricos, los más importantes son los que se muestran a continuación:

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

### 1.3.1 Servicio histórico de variables.

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

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

El término por excepción indica que la variable se envía a los históricos cuando se cumple alguna condición que permite decidir en qué instante debe almacenarse. Dentro de las excepciones más utilizadas por los sistemas SCADA se encuentran [7]:

- ✓ Ejecución de los históricos a una fecha y hora determinada.
- ✓ Envío a los históricos cuando el valor o calidad de la variable cambie.

- ✓ Envío a los históricos cuando se cumple una condición donde pueden intervenir varias variables y estados del sistema. Por ejemplo enviar al histórico la variable1 cuando la variable2 supere un valor determinado.

Los históricos de variables más utilizados son los que se configuran para ser almacenados periódicamente, el tiempo de envío al histórico de una variable depende de la dinámica de la misma y de los requerimientos específicos del proceso. También son utilizados mecanismos para enviar a los históricos sólo datos en forma de sumarios de la variable, lo que permite realizar una especie de compresión de los datos y optimizar los medios persistentes, por ejemplo el promedio en un período de tiempo configurable. Por otro lado, este servicio provee a los clientes la información de las variables en forma de series temporales, opcionalmente sumarizadas por intervalos de tiempo [7].

En la figura 2 se muestra un pequeño esquema donde se evidencian los diferentes módulos que intervienen en el proceso de un historiador. En ella se estampan cinco niveles, expertos en funcionalidades desde la captura de los datos hasta la persistencia de los mismos.

1. **Nivel de campo.** Está compuesto por dispositivos de campo (PLC, RTU, sensores, etc.) que contienen la información del proceso que requiere ser registrado 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 tiempor, BDTR.** Se encarga de recolectar los datos desde el nivel de recolección, procesarlos y servirlos a las aplicaciones de tiempo real, como por ejemplo las tendencias. Además, entrega los datos al nivel de manejo de históricos para garantizar la persistencia de los mismos. Este nivel comúnmente es redundante para garantizar la tolerancia ante fallos.

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

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

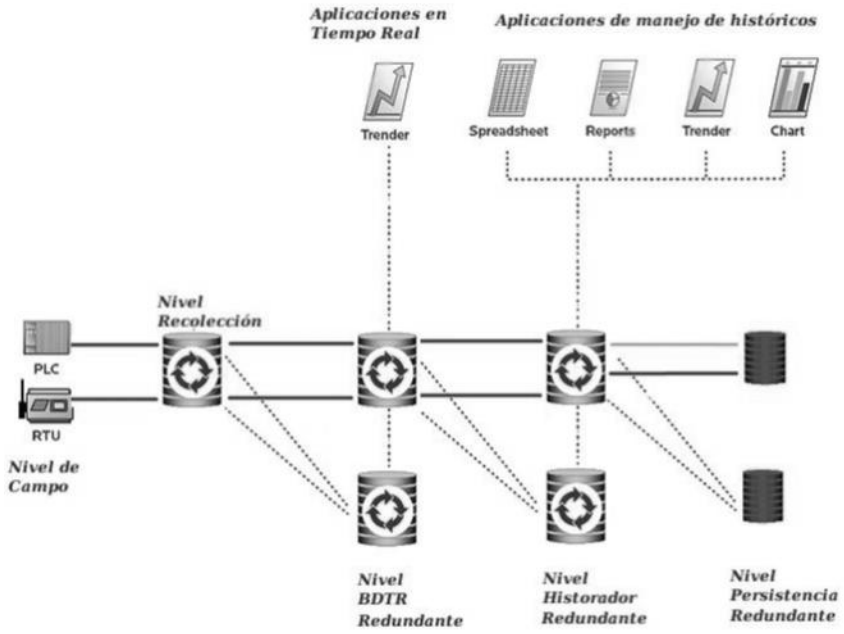



**Figura 2: Esquema de comunicación entre los diferentes módulos que intervienen en el proceso de recolección y archivo de históricos.**

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

En esquemas orientados a objetos, como los presentados en la especificación de históricos HDAIS [8], 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 [7].

### 1.3.2 Servicio histórico de alarmas.

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

Por otro lado, pueden existir alarmas asociadas a los dispositivos de campo del sistema (por ejemplo la falla de un PLC que contiene la información de variables del sistema) y alarmas de sistema (tales como la caída de una red o el fallo catastrófico de recursos de hardware) [7].

Los históricos de alarmas pueden ser divididos en dos partes fundamentales [7]:

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

**Históricos de seguimiento del manejo alarmas**. La ocurrencia de una alarma, indica un hecho que podría llevar a eventos catastróficos. El aviso de la ocurrencia de las alarmas a los operadores, es la acción primaria que se lleva a cabo una vez que estas hayan sido detectadas. Sin embargo, el proceso sucesivo, en la mayoría de los casos, es responsabilidad del operador. Por ejemplo, una vez que el operador es avisado de la ocurrencia de una alarma, el primer paso que debe realizar es reconocer la existencia de la misma, posteriormente debe tomar acciones para restaurar el estado normal del sistema, como por ejemplo la modificación de algún parámetro de control.

### 1.3.3 Servicio histórico de eventos.

Se especializan en el manejo de los eventos que no son considerados catastróficos, pero que tienen impacto en el funcionamiento del sistema y que podrían tornarlo inestable. Dentro de estos se pueden

mencionar autenticaciones, fallas en la ejecución de un comando, caídas temporales de la red, etc. Estos permiten las auditorías de los diferentes servicios del SCADA [7]. Los eventos, se pueden definir como ocurrencias que pueden ser importantes para el análisis del comportamiento y seguridad del sistema. Llevar un registro de eventos, puede ser una herramienta que permite una corrección de los errores de implementación y huecos de seguridad del sistema.

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

- ✓ Eventos de seguridad.
- ✓ Eventos de operatividad.
- ✓ Eventos de comunicación o red.
- ✓ Eventos de configuración.
- ✓ Eventos de seguimiento de comandos.

### 1.3.4 Servicio histórico de bitácoras.

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

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

Reutilizar las experiencias alcanzadas por los operadores, en cuanto al manejo del sistema, detección de errores, exposición de soluciones, entre otros, es uno de los servicios de históricos que se orienta a elevar la calidad operativa del sistema y que contribuye a realizar mejoras basadas en los criterios de los operadores [4].

### 1.3.5 Solicitud de información de los históricos de variables.

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

Las consultas solicitadas por los clientes al servicio de históricos de variables, se definen por un conjunto de parámetros que permiten seleccionar la información de los medios persistentes, dentro de ellos se pueden mencionar los siguientes [9]:

***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. Por lo general si no se especifica este parámetro, se considera que son todos los valores hasta el tiempo actual.

***Bounds:*** Por defecto las consultas toman todos los valores entre el *StartTime* y *EndTime*, si este parámetro está activo, indica que se tomarán todos los datos incluidos en el intervalo incluso aquellos que se corresponden con el *EndTime*, de otra forma no se incluirán los valores que se corresponden con *EndTime*. Si este parámetro está activo y no existe un valor que se corresponda exactamente con el *StartTime*, se tomará el primer valor que se encuentre anterior al *StartTime*. De igual forma si no existe un valor exactamente en el *EndTime*, se ofrecerá el primer valor que se encuentre después del *EndTime*.

***Aggregate:*** Este parámetro está inspirado en la especificación OPC HDA [9], básicamente define los métodos para realizar sumarios de la información del punto. Lo agregados más utilizados son: promedio, mínimo y máximo en un rango determinado. Solamente se permite el cálculo de agregados sobre valores numéricos y se omiten los valores con calidad mala. En el tema de calidad, en dependencia de la aplicación del servidor, se pueden incluir los valores con calidad incierta.

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

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

## 1.4 Sistemas de gestión de base de datos.

### 1.4.1 Base de datos PostgreeSQL.

#### *1.4.1.1 Descripción.*

PostgreeSQL es un motor de base de datos relacionales que verifica integridad referencial con gran funcionalidad como base de datos, aunque un poco más lenta que otros motores. Distribuido bajo licencia BSD (*Berkeley Software Distribution*) y con su código fuente disponible libremente. Es el sistema de gestión de base de datos de código abierto más potente del mercado. (Sitio oficial de PostgreeSQL). PostgreeSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multi-hilos para garantizar la estabilidad del sistema.

#### *1.4.1.2 Características.*

- ✓ Acceso por la red mediante el protocolo SSL
- ✓ Documentación completa.
- ✓ Licencia BSD.
- ✓ Posee funciones y procedimientos almacenados en numerosos lenguajes de programación, entre otros PL/pgSQL.
- ✓ Tiene numerosos tipos de datos y posibilidad de definir nuevos tipos. Además de los tipos estándares en cualquier base de datos, entre otros, tipos geométricos, de direcciones de red, de cadenas binarias, UUID, XML, matrices, etc.
- ✓ Soporta el almacenamiento de objetos binarios grandes (gráficos, videos, sonido, etc.)
- ✓ APIs para programar en C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, PHP, Lisp, Scheme, Qt. y muchos otros.

- ✓ PostgreeSQL está basado en lenguajes SQL 92, SQL99, SQL2003, SQL2008. Incluyendo:
  - Llaves primarias (primary keys) y ajenas (foreign keys).
  - Check, unique y not null constraints.
  - Columnas auto-incrementales.
  - Índices compuestos, únicos, parciales y funcionales en cualquiera de los métodos de almacenamiento disponibles, B-tree, R-tree, hash ó Gist.
  - Sub-selects, consultas recursivas.
  - Joins.
  - Vistas (views).
  - Disparadores (triggers).
  - Reglas (Rules).
  - Herencia de tablas (Inheritance).
  - Eventos LISTEN/NOTIFY.

#### *1.4.1.3 Limitaciones.*

| **Límite** | **Valor** |
|---|---|
| Tamaño máximo de base de dato | Ilimitado (Depende de tu sistema de almacenamiento) |
| Tamaño máximo de tabla | 32 TB |
| Tamaño máximo de fila | 1.6 TB |
| Tamaño máximo de campo | 1 GB |
| Máximo número de filas por tabla | Ilimitado |
| Máximo número de columnas por tabla | 250 - 1600 (dependiendo del tipo) |
| Máximo número de índices por tabla | Ilimitado |

**Tabla 1: Limitaciones de PostgreeSQL.**

### 1.4.2 Base de datos SQLite.

### *1.4.2.1 Descripción.*

SQLite es una biblioteca escrita en C que implementa un motor de base de datos SQL empotrado. Es el mayor despliegue del motor de base de datos SQL en el mundo. El código fuente de SQLite es en el dominio público.

### *1.4.2.2 Características.*

- ✓ El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo fichero estándar.
- ✓ Posee soporte transaccional.
- ✓ Es muy rápida y tiene un escaso tamaño (aproximadamente 25 mil líneas de código C). Alta portabilidad.
- ✓ El tamaño que ocupa el SQLite en memoria es inferior a 250 KB.
- ✓ SQLite permite varios lectores simultáneos y un solo escritor.
- ✓ La base de datos puede tener múltiples procesos ejecutándose al mismo tiempo.
- ✓ Posee falta de tipos de datos definidos en SQL. Por defecto todos los datos se almacenan en cadenas de caracteres acabadas en nulo, a excepción de las columnas definidas como INTEGER PRIMARY KEY que se almacenan como enteros y hacen las veces de campo auto-incremental. Para suplir esta carencia se utiliza la directiva pysqlite_pragma expected_types utilizando la librería pysqlite.
- ✓ Permite crear varias tablas y establecer relaciones entre ellas utilizando toda la potencia del lenguaje SQL 92.
- ✓ SQLite define las API que permite al usuario agregar nuevas funciones de SQL.
- ✓ Cuenta con una utilidad que permite ejecutar comandos SQL contra una base de datos SQLite, crear la base de datos, realizar consultas, insertar datos, etc.
- ✓ Permite extender su funcionalidad, utilizando toda la potencialidad de los lenguajes Python, C, C++.
- ✓ La base de datos alcanza un tamaño de hasta 2 terabytes.
- ✓ Campos de longitud indefinida.

- ✓ A diferencia de otros motores de base de datos empotrados SQLite si permite su uso de forma compartida por varios usuarios, siendo una opción muy interesante en pequeños entornos de trabajo.
- ✓ Permite crear vistas y triggers.
- ✓ Bloquea toda la base de datos y las tablas cuando una operación está realizándose, esto es conocido como “bloqueo de base de datos de nivel”.

#### *1.4.2.3 Limitaciones.*

- ✓ No permite crear procedimientos almacenados [10].
- ✓ Las vistas son de solo lectura, el usuario no puede ejecutar un DELETE, INSERT o UPDATE en una vista. Pero se puede crear un triggers que se active en un intento de DELETE, INSERT, UPDATE en una vista y hacerlo en las sentencias del triggers.
- ✓ SQLite no tiene implementado el operador LIKE.
- ✓ Los comandos GRANT and REVOKE no están implementados, pues como comúnmente funcionan en aplicaciones cliente/servidor no cumple objetivo ejecutarlos en base de datos empotradas. (Sitio Oficial de SQLite)
- ✓ Las llaves foráneas están deshabilitadas pero a partir de la versión 3.6.19 se pueden activar a partir de un procedimiento descrito el sitio oficial de SQLite.

### 1.4.3 Base de datos Derby.

#### *1.4.3.1 Descripción.*

Apache Derby es una base de datos relacional de código abierto, y parte del proyecto Apache DB, está escrito en java. Derby soporta datos de almacenamiento y de interrogación SQL, proporcionando así todas las funciones que normalmente poseen los grandes sistemas de base de datos. Además puede correr en modo cliente/servidor al igual que los grandes motores de base de datos y tiene un tamaño de 2 mb de espacio en disco. Actualmente se distribuye como Sun Java DB y su última versión estable es la 10.8.2.2.

### *1.4.3.2 Características.*

- ✓ Presenta APIs para JDBC (*Java Database Connectivity*) y SQL. Soporta todas las características de SQL92 y la mayoría de SQL99.
- ✓ Su código mide alrededor de 2000KB comprimido, aunque ocupa poco espacio representa 8 veces más que el espacio que ocupa el SQLite.
- ✓ Soporta cifrado completo, roles y permisos. Además posee SQL SCHEMAS para separar la información en una única base de datos y garantizar el control completo de los usuarios.
- ✓ Soporta internamente cifrado y compresión.
- ✓ Trae soporte multilenguaje y localizaciones específicas.
- ✓ Transacciones y recuperación ante errores ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad)
- ✓ Derby sólo permite un único proceso para tener la base de datos abierta al mismo tiempo en el modo integrado.
- ✓ Derby provee un driver JDBC que permite embeber a Derby en cualquier solución Java.
- ✓ Derby soporta una amplia variedad de tipos de datos, incluyendo XML.
- ✓ Posee tres productos asociados a la marca:
  - *Derby Embedded Database Engine*: El motor propiamente dicho.
  - *Derby Network Server*: Permite convertir Derby en una base de datos que sigue el modelo cliente-servidor tradicional.
  - *Database Utilities*: Un paquete de utilidades.
- ✓ Características de los esquemas.

| **Esquema** | **Navegación** | **Edición Visual** | **Secuencias de Comandos DDL** |
|---|---|---|---|
| Tablas | X | X | X |
| Vistas | X | X | X |
| Sinónimos | X | X | X |
| Índices | X | X | X |
| Disparadores | X | X | X |

| | | | |
|---|---|---|---|
| Procedimientos | X | X | X |
| Funciones | X | X | X |

**Tabla 2: Características de los esquemas.**

- ✓ Derby una de las pocas base de datos de este estilo que permite usar Procedimientos Almacenados con más de un parámetro de salida (output).

#### *1.4.3.3 Limitaciones.*

Derby es utilizable sólo por Java y lenguajes de script que se ejecutan en la máquina virtual de Java (Jython, JRuby, Jacl, etc.) y en la actualidad sólo se expone a través del controlador JDBC, hay un controlador ODBC para el Derby, pero ya no se mantiene.

- ✓ No permite el ALTER DOMAIN.
- ✓ No realiza referencias a tablas locales.
- ✓ No permite la actualización completa de los cursores.
- ✓ Las vistas son de solo lectura, no se le puede realizar actualizaciones.

### 1.4.4 Base de datos Berkeley.

#### *1.4.4.1 Descripción.*

La Base de Datos Berkeley es un motor de base de datos que provee a los desarrolladores una base de datos simple, rápida y segura, con cero administración, debido a que funciona como una biblioteca que se enlaza directamente en la aplicación eliminando la penalización en el rendimiento de los sistemas cliente-servidor y el procesamiento SQL, ideal para consultas estáticas sobre datos dinámicos. Berkeley DB pertenece y es desarrollado por la compañía Sleepycat Software. Es multiplataforma, está disponible con código fuente y licencia de libre distribución (free software).

#### *1.4.4.2 Características.*

- ✓ Soporta multiprocesos.
- ✓ Alta fiabilidad y disponibilidad.

- ✓ Recuperación de datos en forma secuencial e indexada.
- ✓ Procesos múltiples por aplicación e hilos múltiples por proceso.
- ✓ Encriptación de datos por el algoritmo AES.
- ✓ Registros de hasta 4GB y tablas de hasta 256TB.
- ✓ Cumple al 100% con ACID.
- ✓ Respaldos en frio y en caliente.
- ✓ Alta fiabilidad y disponibilidad. Su completa semántica transaccional garantiza la integridad de los datos, recuperación ante fallos y replicación.
- ✓ Administración automática, toda la administración se realiza vía API, ocultándose al usuario final.
- ✓ Soporte de los lenguajes C, C++, Java, Perl, Python, PHP, TCL y Ruby.
- ✓ Disponible en los sistemas operativos: Linux, Windows, BSD Unix, Solaris, Mac OS X, etc.
- ✓ Gran probabilidad de sobrevivir a fallos del sistema y ser capaces de realizar COMMIT y ROLLBACK en transacciones.
- ✓ La PRIMARY KEY es más rápida que cualquier otro índice, pues se almacena junto con los datos.

#### *1.4.4.3 Limitaciones.*

- ✓ Abrir muchas tablas a la vez resulta lento.
- ✓ El escaneo secuencial es más lento ya que los datos en tablas se almacenan en B-trees y no en un fichero de datos separado.
- ✓ Si se eliminan ficheros de log antiguos que estén en uso, Berkeley DB no es capaz de recuperar todo y puede perder datos si algo no va bien.

### 1.4.5 Selección del sistema gestor de base de datos.

Analizando las descripciones, características y limitaciones de las base de datos empotradas expuestas anteriormente, teniendo en cuenta las funcionalidades del gestor de base de datos PostgreeSQL y la necesidad de utilizar un sistema gestor de base de datos empotrado. Se arriba a la conclusión de que el sistema de base de datos empotrado más factible para su uso en el módulo es Berkeley DB.

Este sistema de base de datos empotrado es totalmente libre, de código abierto y multiplataforma. Permite la utilización de vistas, procedimientos almacenados, comandos DDL y DML. Además permite la

conectividad con el lenguaje C++ y el uso de lenguajes scripts al estilo de PL/SQL de Oracle.

## 1.5 Bibliotecas gráficas de desarrollo.

### 1.5.1 Qt.

Qt. es una biblioteca multiplataforma, software libre y de código abierto que es utilizada para desarrollar interfaces gráficas de usuarios y también para el desarrollo de aplicaciones sin interfaz gráfica como herramientas de consola y servidores. Proporciona la mayoría de las entidades gráficas que verá empleadas en una aplicación KDE: menús, botones, regletas, etc. Es producido por la división de software Qt. de Nokia, que entró en vigor después de la adquisición por parte de Nokia de la empresa noruega Trolltech.

Qt. es utilizada fundamentalmente en entornos de escritorio KDE, aunque también es perfectamente funcional en otros escritorios como GNOME, Windows y Mac OS. Hace uso del lenguaje de programación C++ de forma nativa, aunque actualmente es posible utilizarlo en otros lenguajes de programación a través de bindings (adaptación de una biblioteca para ser utilizada en otro lenguaje de programación distinto de aquél en el que ha sido creada). Funciona en las principales plataformas y tiene un amplio apoyo.

### 1.5.2 GTK+.

GTK+ son las siglas de GIMP Toolkit, es una biblioteca que permite crear interfaces gráficas de usuario, se distribuye bajo la licencia pública general (GPL), lo que hace de GTK+ un producto completamente libre. (GTK, 2007).

GTK+ es multi-plataforma, se ha extendido hasta Microsoft Windows y muchos derivados de Unix, como Linux y Mac OS. Está escrita en C y tiene extensiones en varios lenguajes como C++, Python, Perl y muchos más. Se empleó inicialmente en el proyecto *Gnu Image Manipulation Program ToolKit*, por eso su nombre: GTK+, se ha extendido rápidamente por su estabilidad y una de las implementaciones que más se ha usado es la de C++, llamada Gtkmm (2007).

Gtkmm es la implementación oficial de GTK+ escrita en C++, le adiciona a GTK+ las potencialidades del paradigma orientado a objetos, la herencia para crear nuevos componentes, polimorfismo, manejo de memoria para construir y destruir objetos, elimina el uso de las macros de C y muchas otras mejoras.

## 1.6 Tecnologías y herramientas a utilizar para el desarrollo.

Para la selección de las tecnologías y herramientas a utilizar en el desarrollo se tuvieron en cuenta una serie de requisitos como: su aceptación a nivel internacional, que fueran de libre distribución y fueran además, las más adecuadas a los requerimientos del sistema a desarrollar. A continuación se enumeran y describen cada una.

### 1.6.1 Microsoft Visual C++ 2008 como IDE de desarrollo.

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

A la hora de elegir Visual Studio en especial Visual C++ como IDE de desarrollo influyó que este permite la programación en C++, presenta buen completamiento de código y brinda facilidades en la vinculación con la biblioteca grafica Qt. Además de que Visual C++ 2008 fue la herramienta empleada para el desarrollo de la mayor parte del SCADA Ligero por lo que era fundamental la utilización de este IDE para tener una correcta vinculación con la librería a desarrollar.

### 1.6.2 Qt. como biblioteca gráfica.

A partir del estudio teórico realizado se optó por desarrollar la interfaz gráfica con la biblioteca Qt., debido a que la interfaz de programación de la misma está totalmente orientada a objetos y tienen una buena respuesta y un uso adecuado de la memoria. Además de que dicha biblioteca es multiplataforma y presenta una licencia LGPL que nos permite cerrar el código y poder venderlo y es libre para su uso en sistemas.

También tiene ventaja sobre otros frameworks o toolkits que proveen un recubrimiento para C++, debido a que está implementada directamente en C++. Hay que destacar que la selección de esta biblioteca se debe en gran medida también a la amplia comunidad de desarrollo que le brinda soporte y porque la misma posee muchas funcionalidades de alto rendimiento.

El centro decidió que las interfaces graficas fueran desarrolladas en Qt., además de que Qt. posee un amplio conjunto de widgets estándares y permite desarrollar controles personalizados. La calidad en tiempo de ejecución es alta y no depende de ninguna otra biblioteca, aparte de sí misma.

### 1.6.3 C++ como lenguaje de programación.

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

### 1.6.4 Visual Paradigm como herramienta CASE.

Las herramientas CASE (***C****omputer* ***A****ided* ***S****oftware* ***E****ngineering*, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero. Estas herramientas pueden ayudar a ingenieros, desarrolladores en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores, realización de ingeniería inversa entre otras funciones.

Para el desarrollo del software se utilizó Visual Paradigm, la cual como herramienta UML profesional soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue, incluye además actividades como la gestión de proyectos y la estimación. Contribuye con la rápida construcción de aplicaciones de calidad mejores y a un menor coste, es fácil de usar y permite dibujar todos los tipos de diagramas de clases, código inverso, generar código

desde diagramas y generar documentación. Apoya un conjunto de lenguajes tanto en la generación del código como en la Ingeniería Inversa por ejemplo Java, C + +, CORBA IDL, PHP, XML Schema, Ada y Python.

### 1.6.5 UML como lenguaje de modelado.

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas más conocido y utilizado en la actualidad; está respaldado por OMG (*Object Management Group*). Es un lenguaje grafico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de base de datos y componentes de software reutilizables.

### 1.6.6 RUP como metodología de desarrollo.

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

El Proceso Unificado de Rational (*Rational Unified Process* en inglés, habitualmente resumido como RUP) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado (UML), constituye la metodología estándar más utilizada para el análisis y diseño, implementación y documentación de sistemas orientados a objetos.

RUP divide el proceso de desarrollo en ciclos, teniendo un producto final al terminar cada ciclo, estos se dividen en fases (inicio, elaboración, construcción y transición) que finalizan con un hito donde se debe tomar una decisión importante.

Se caracteriza por ser:

- ✓ **Dirigido por casos de uso:** Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos.
- ✓ **Centrado en la arquitectura:** La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo.
- ✓ **Iterativo e Incremental:** RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros.

### 1.6.7 Berkeley como sistema de base de datos embebido.

El motor de base de datos Berkeley proporciona una interfaz para el manejo de base de datos SQL y una herramienta para su tratamiento a través de la línea de comandos. No necesita administración y garantiza un alto rendimiento por la flexibilidad que posee para personalizar la base de datos a los requerimientos del sistema que lo utilice. Permite ser accedido desde el lenguaje C++. Soporta múltiples conexiones por lo que es la más indicada para el uso en el módulo. Como muestra de su gran desempeño es usado por empresas de renombrado prestigio como: Airbus, Amazon, AOL, Cisco Systems, eBay, EMC, Google, Hitachi, HP, Motorola, Nortel, RSA Security, San, entre otras. Por las características anteriormente expuestas, es la base de datos empotrada que más se ajusta a las necesidades del módulo de gestión y archivo de datos del SCADA Ligero.

## 1.7 Consideraciones parciales del capítulo.

Con los elementos estudiados en este capítulo se muestran los distintos servicios que generalmente brinda un módulo de gestión y archivo de datos. Además se expusieron algunas características de algunos SGBD empotrados, ventajas y particularidades a tener en cuenta al seleccionarlos según las necesidades. Este capítulo deja sentada las bases para un mayor entendimiento de las características del sistema las cuales serán expuestas en el siguiente capítulo.

# CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA.

## 2.1 Introducción.

El presente capítulo muestra la descripción del sistema propuesto. En el mismo se exponen los requisitos funcionales y no funcionales que rigen el desarrollo de la solución propuesta. Partiendo de esto, se determinan los casos de uso y se describen los procesos de las principales funcionalidades del sistema.

## 2.2 Soluciones técnicas.

Para darle cumplimiento al objetivo de este trabajo se pretende desarrollar un módulo de gestión y archivo de datos para el SCADA Ligero. Integrado por una biblioteca estática para el almacenamiento de información usando un SGBD embebido y una aplicación independiente al sistema que muestre dicha información.

## 2.3 Modelo del dominio.

Un modelo del dominio captura los tipos más importantes de los objetos en contexto del sistema. Los objetos del dominio representan las “cosas” que existen o los eventos que suceden en el entorno en el que trabaja el sistema [11].

En el presente epígrafe se describen los conceptos más importantes y sus relaciones, para lograr un mejor entendimiento de la solución.

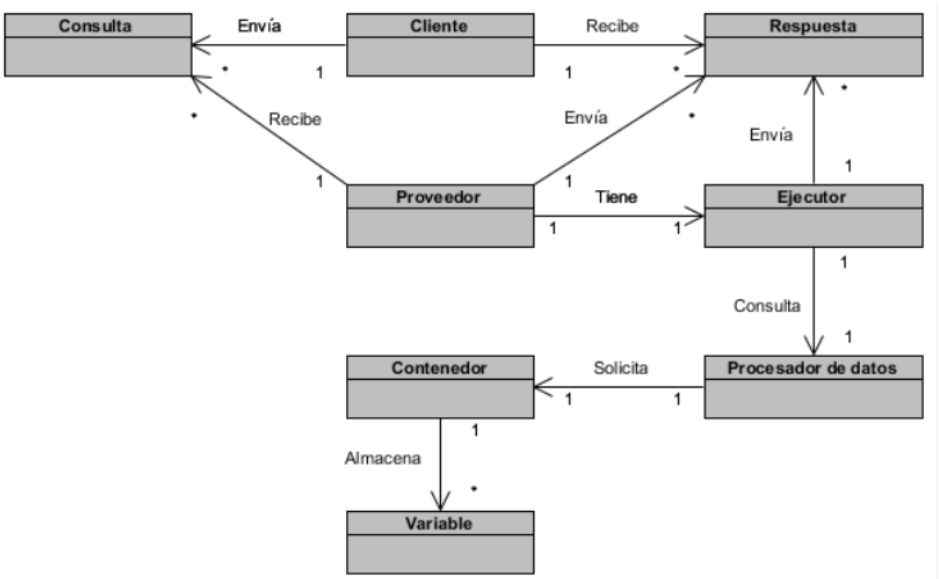



**Figura 3: Modelo de dominio.**

### 2.3.1 Clases conceptuales o glosario de términos del dominio.

- ✓ **Consulta:** Solicitud de datos de los históricos.
- ✓ **Respuesta:** Resultado de la ejecución de una consulta o mensaje de error.
- ✓ **Cliente:** Encargado de llevar a cabo una solicitud de consulta y luego procesar la respuesta recibida.
- ✓ **Proveedor:** Gestiona las solicitudes de consultas a la base de datos. 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).
- ✓ **Procesador de datos:** Contiene la lógica de almacenamiento y acceso a los datos históricos que persisten en la base de datos.
- ✓ **Contenedor:** Convierte los datos genéricos recibidos y los transforma a datos históricos para luego almacenarlos temporalmente hasta ser solicitados por el procesador de datos.
- ✓ **Variable:** Representa una estructura de datos definida para su almacenamiento histórico.

## 2.4 Descripción del sistema propuesto.

El sistema propuesto se estructura en dos partes fundamentales: una biblioteca estática para el almacenamiento de los datos y una aplicación visual para mostrar la información almacenada. La biblioteca se construye usando para ello el lenguaje C++ y un gestor de base de datos empotrado. Esta debe implementar una interfaz que permita la interacción desde el sistema (SCADA Ligero), encapsulando funcionalidades para almacenar en una base de datos los valores de las variables y devolver dichos valores almacenados, así como definir un recolector de basura que se encargará de eliminar de la base de datos toda la información que lleve más tiempo del definido que debe persistir dicha información.

La aplicación visual es la encargada de mostrar dichos valores almacenados en un rango de tiempo definido por el operador, además de unos agregados que permitirán al operador un mejor entendimiento de dicha información almacenada. La aplicación debe permitir al operador ver la gráfica de tendencia de una variable en un intervalo de tiempo definido por él.

### 2.4.1 Requisitos funcionales.

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

El sistema debe permitir:

**RF1:** Almacenar variables.

Este requisito expresa la necesidad de almacenar todos los puntos que maneja el SCADA Ligero.

**RF2:** Definir recolector de basura.

Este requisito expresa la necesidad de mantener el tiempo requerido en la base de datos los puntos que sean necesarios para su posterior análisis.

**RF3:** Devolver variables almacenadas.

Este requisito expresa la necesidad de devolver los puntos almacenados en la base de datos.

**RF4:** Mostrar variables en un intervalo de tiempo.

Este requisito expresa la necesidad de devolver los puntos almacenados en la base de datos en un intervalo de tiempo.

**RF5:** Mostrar tendencia de una variable en un intervalo de tiempo.

Este requisito expresa la necesidad de mostrar en una gráfica la tendencia de una variable en un intervalo de tiempo.

**RF6:** Mostrar mínimo valor de una variable en un intervalo de tiempo.

Este requisito expresa la necesidad de mostrar el mínimo valor de un punto almacenado en la base de datos en un intervalo de tiempo.

**RF7:** Mostrar máximo valor de una variable en un intervalo de tiempo.

Este requisito expresa la necesidad de mostrar el máximo valor de un punto almacenado en la base de datos en un intervalo de tiempo.

**RF8:** Mostrar promedio de una variable en un intervalo de tiempo.

Este requisito expresa la necesidad de mostrar el valor promedio de un punto almacenado en la base de datos en un intervalo de tiempo.

### 2.4.2 Requisitos no funcionales.

Las características o requisitos no funcionales (RNF) del sistema, son propiedades o cualidades que el producto debe tener. Son las características que hacen al producto atractivo, usable, rápido y confiable. Estos requisitos son muy importantes para que los clientes y usuarios puedan valorar características no funcionales del software como: seguridad, confiabilidad y usabilidad.

Los requisitos no funcionales que debe cumplir son:

**Restricciones en el diseño y la implementación.**

**RNF 1:** Uso del framework Qt. para el desarrollo de la interfaz visual.

**RNF 2:** Emplear el paradigma programación orientado a objetos.

**RNF 3:** C++ como lenguaje de programación.

**Requisitos de Seguridad.**

**RNF 4:** Confidencialidad: La información manejada por el sistema deberá estar protegida de acceso no autorizado y divulgación.

**Requisitos de Usabilidad.**

**RNF 5:** Las operaciones de la aplicación visual deben ser lo más intuitivas posibles a la vista del cliente.

**Requisitos de Soporte.**

**RNF 6:** Una vez terminada la aplicación debe ser de fácil instalación y puesta en ejecución.

**RNF 7:** Licencia bajo código abierto: El sistema debe cumplir con los lineamientos necesarios para la producción de software libre y la comunidad de desarrollo y soporte.

**Requisitos Rendimiento.**

**RNF 8:** La velocidad de publicación debe ser igual a la del acceso de los datos. La velocidad debe ser menor a 100 milisegundos (ms).

### 2.4.3 Actores del sistema.

Actor del sistema es toda entidad externa al sistema que guarda una relación con éste y que le demanda una funcionalidad. Esto incluye a los operadores humanos pero también incluye a todos los sistemas externos, además de entidades abstractas, como el tiempo.

| Actor | Descripción |
|---|---|

| | |
|---|---|
| Operador | Es la persona que se encarga de iniciar la conexión, devolver los datos almacenados, supervisar los valores de las variables almacenadas y cerrar la conexión. |
| Sistema | Es el sistema que se encarga del almacenamiento de los datos y la definición del recolector de basura. |

**Tabla 3: Descripción de los actores.**

### 2.4.4 Casos de uso del sistema.

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

| Críticos | Secundarios | Auxiliares |
|---|---|---|
| -Almacenar variable.<br>-Definir recolector de basura.<br>-Devolver variables almacenadas.<br>-Mostrar variables en un intervalo de tiempo.<br>-Mostrar tendencia de una variable en un intervalo de tiempo. | - Mostrar agregados. | No aplica |

**Tabla 4: Clasificación de los casos de uso del sistema.**

### 2.4.5 Diagramas de los casos de uso del sistema.

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

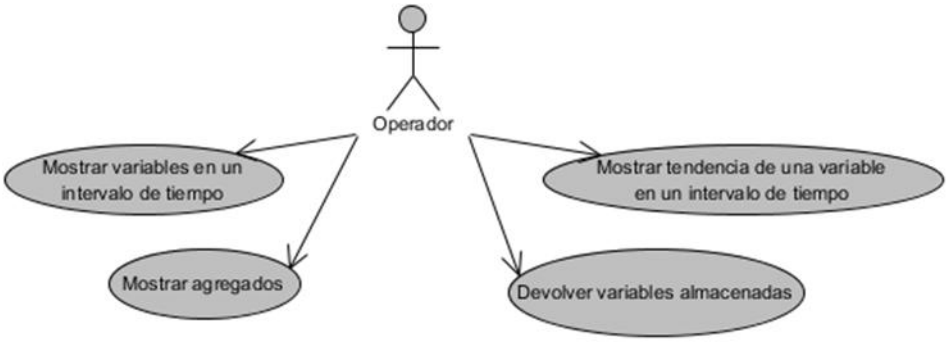
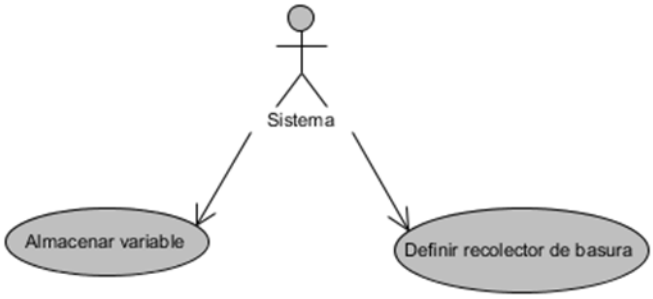



**Figura 4: Diagrama de caso de uso del sistema**




**(operador).**

**Figura 5: Diagrama de caso de uso del sistema (sistema).**

### 2.4.6 Descripción de los casos de uso del sistema.

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

#### *2.4.6.1 RF Almacenar variable.*

| | |
|---|---|
| **Nombre del CU** | Almacenar variable |
| **Objetivo** | El operador tiene como objetivo almacenar los valores de los puntos que maneja. |
| **Actores** | Sistema(Inicia) |

| | |
|---|---|
| **Resumen** | El caso de uso se inicia cuando el sistema comienza a almacenar los puntos que está controlando. Todos estos valores son almacenados en una base de datos previamente definida. El caso de uso termina cuando se almacenan los puntos que se están controlando. |
| **Complejidad** | Alta |
| **Prioridad** | Crítico |
| **Precondiciones** | El sistema debe definir cómo se va a llamar la base de datos y donde debe estar ubicada. |
| **Pos condiciones** | Los puntos quedaron almacenados para su posterior análisis. |

| **Flujo de eventos** | |
|---|---|
| **Flujo básico** | |
| **Actor** | **Sistema** |
| 1. Envía una variable para almacenar. | 2. Inicia la conexión. |
| | 3. Verifica si existe la tabla. En caso de que exista (Ver flujo alterno 1). |
| | 4. Crea la tabla. |
| | 5. Verifica que no ocurra un error. En caso de que exista uno (Ver flujo alterno 2). |
| | 6. Inserta la variable en la base de datos. |
| | 7. Termina el caso de uso |
| **Flujos alternos** | |
| **Nº1. La tabla existe.** | |
| **Actor** | **Sistema** |
| | 1. Retorna al paso 5. |
| **Nº2. Ocurre un error al crear o abrir la tabla.** | |
| **Actor** | **Sistema** |
| | 1. Muestra un mensaje de error diciendo “No se pudo crear la tabla”. |

| | | |
|---|---|---|
| | 2. Retorna al paso 7. | |
| **Relaciones** | **CU Incluidos** | No tiene. |
| | **CU Extendidos** | No tiene. |
| **Requisitos funcionales** | RF 1 | |

**Tabla 5: Descripción del caso de uso "Almacenar variable".**

### *2.4.6.2 RF Definir recolector de basura.*

| | | |
|---|---|---|
| **Nombre del CU** | Definir recolector de basura | |
| **Objetivo** | El sistema tiene como objetivo definir el tiempo que debe estar almacenada la información. | |
| **Actores** | Sistema(Inicia) | |
| **Resumen** | El caso de uso se inicia cuando el sistema desea definir el tiempo que debe estar almacenada la información. El caso de uso termina cuando cierra la aplicación. | |
| **Complejidad** | Alta | |
| **Prioridad** | Crítico | |
| **Precondiciones** | Deben existir puntos almacenados. | |
| **Pos condiciones** | Se definió el tiempo que debe estar almacenada la información | |
| **Flujo de eventos** | | |
| **Flujo básico** | | |
| **Actor** | | **Sistema** |
| 1. Definir el tiempo que debe estar almacenada la información. | | 2. Inicia la conexión. |
| | | 3. Elimina la información que no está enmarcada en el tiempo definido. |
| | | 4. Termina el caso de uso |
| **Flujos alternos** | | |

| | | |
|---|---|---|
| **No tiene.** | | |
| **Relaciones** | **CU Incluidos** | No tiene. |
| | **CU Extendidos** | No tiene. |
| **Requisitos funcionales** | RF 2 | |

**Tabla 6: Descripción del caso de uso "Definir recolector de basura".**

### *2.4.6.3 RF Devolver variables almacenadas.*

| | | |
|---|---|---|
| **Nombre del CU** | Devolver variables almacenadas | |
| **Objetivo** | El operador tiene como objetivo devolver los puntos almacenados. | |
| **Actores** | Operador (Inicia) | |
| **Resumen** | El caso de uso se inicia cuando el operador desea devolver los puntos que están almacenados. Todos estos valores se muestran en una tabla. El caso de uso termina cuando se devuelven los puntos almacenados. | |
| **Complejidad** | Alta | |
| **Prioridad** | Crítico | |
| **Precondiciones** | Deben existir puntos almacenados. | |
| **Pos condiciones** | Se devolvieron las variables almacenadas | |
| **Flujo de eventos** | | |
| **Flujo básico** | | |
| **Actor** | | **Sistema** |
| 1. Solicita las variables almacenadas. | | 2. Inicia la conexión. |
| | | 3. Devuelve las variables almacenadas. |
| | | 4. Verifica que no ocurra un error. En caso de que exista uno (Ver flujo alterno 1). |
| | | 5. Termina el caso de uso |

| **Flujos alternos** | | |
|---|---|---|
| **Nº1. Ocurre un error al devolver las variables almacenadas.** | | |
| **Actor** | **Sistema** | |
| | 1. Muestra un mensaje de error “No existe la tabla”.<br>2. Retorna al paso 5. | |
| **Relaciones** | **CU Incluidos** | No tiene. |
| | **CU Extendidos** | No tiene. |
| **Requisitos funcionales** | RF 3 | |

**Tabla 7: Descripción del caso de uso “Devolver variables almacenadas”.**

### *2.4.6.4 RF Mostrar variables en un intervalo de tiempo.*

| **Nombre del CU** | Mostrar variables en un intervalo de tiempo |
|---|---|
| **Objetivo** | El operador tiene como objetivo mostrar las variables almacenadas en un intervalo de tiempo. |
| **Actores** | Operador (Inicia) |
| **Resumen** | El caso de uso se inicia cuando el operador desea mostrar las variables almacenadas en un intervalo de tiempo. El sistema busca los puntos almacenados en ese intervalo de tiempo. El caso de uso termina cuando se muestran los puntos almacenados en el intervalo de tiempo. |
| **Complejidad** | Alta |
| **Prioridad** | Crítico |
| **Precondiciones** | Deben existir los puntos almacenados. |
| **Pos condiciones** | Se mostraron las variables almacenadas en un intervalo de tiempo. |
| **Flujo de eventos** | |
| **Flujo básico** | |

| **Actor** | **Sistema** |
|---|---|
| 1. Introduce el intervalo de tiempo. | 2. Verifica que el intervalo este correcto. En caso de que exista uno (Ver flujo alterno 1). |
| | 3. Devuelve las variables almacenadas en ese intervalo de tiempo. |
| | 4. Verifica que no ocurra un error. En caso de error (Ver flujo alterno 2). |
| | 5. Muestra las variables almacenadas en un intervalo de tiempo. |
| | 6. Termina el caso de uso |
| **Flujos alternos** | |
| **Nº1. Ocurre un error al introducir el intervalo de tiempo.** | |
| **Actor** | **Sistema** |
| | 1. Muestra un mensaje de error "Debe introducir correctamente el intervalo de tiempo".<br>2. Retorna al paso 1. |
| **Nº2. Ocurre un error al devolver las variables almacenadas en ese intervalo de tiempo.** | |
| **Actor** | **Sistema** |
| | 1. Muestra un mensaje de error "No existe la tabla".<br>2. Retorna al paso 6. |
| **Relaciones** | **CU Incluidos** No tiene. |
| | **CU Extendidos** No tiene. |
| **Requisitos funcionales** | RF 4 |

**Tabla 8: Descripción del caso de uso "Mostrar variables en un intervalo de tiempo".**

### *2.4.6.5 RF Mostrar tendencia de una variable en un intervalo de tiempo.*

| **Nombre del CU** | Mostrar tendencia de una variable en un intervalo de tiempo. |
|---|---|

| | |
|---|---|
| **Objetivo** | El operador tiene como objetivo mostrar la tendencia de una variable en un intervalo de tiempo. |
| **Actores** | Operador(Inicia) |
| **Resumen** | El caso de uso se inicia cuando el operador desea mostrar la tendencia de una variable en un intervalo de tiempo. El sistema busca los puntos almacenados en ese intervalo de tiempo y muestra los distintos valores que tenía en ese intervalo en una gráfica. El caso de uso termina cuando se muestra la tendencia de un punto en un intervalo de tiempo |
| **Complejidad** | Alta |
| **Prioridad** | Crítico |
| **Precondiciones** | Deben existir los puntos almacenados. |
| **Pos condiciones** | Se mostró la tendencia de una variable en un intervalo de tiempo. |

| **Flujo de eventos** | |
|---|---|
| **Flujo básico** | |
| **Actor** | **Sistema** |
| 1. Introduce el intervalo de tiempo | 2. Verifica que el intervalo este correcto. En caso de que exista uno (Ver flujo alterno 1). |
| | 3. Devuelve las variables almacenadas en ese intervalo de tiempo. |
| | 4. Verifica que no ocurra un error. En caso de error (Ver flujo alterno 2). |
| 5. Introduce el identificador de la variable. | 6. Muestra la tendencia de la variable en un intervalo de tiempo. |
| | 7. Termina el caso de uso |

| **Flujos alternos** | |
|---|---|
| **Nº1. Ocurre un error al introducir el intervalo de tiempo.** | |
| **Actor** | **Sistema** |
| | 1. Muestra un mensaje de error “Debe introducir |

| | | |
|---|---|---|
| | correctamente el intervalo de tiempo".<br>2. Retorna al paso 1. | |
| **Nº2. Ocurre un error al devolver las variables almacenadas en ese intervalo de tiempo.** | | |
| **Actor** | **Sistema** | |
| | 1. Muestra un mensaje de error "No existe la tabla".<br>2. Retorna al paso 7. | |
| **Relaciones** | **CU Incluidos** | No tiene. |
| | **CU Extendidos** | No tiene. |
| **Requisitos funcionales** | RF 5 | |

**Tabla 9: Descripción del caso de uso "Mostrar tendencia de una variable en un intervalo de tiempo".**

### *2.4.6.6 RF Mostrar agregados.*

| | |
|---|---|
| **Nombre del CU** | Mostrar agregados. |
| **Objetivo** | El operador tiene como objetivo mostrar el mínimo valor, el máximo valor o el promedio de una variable en un intervalo de tiempo. |
| **Actores** | Operador(Inicia) |
| **Resumen** | El caso de uso se inicia cuando el operador desea mostrar el mínimo valor, el máximo valor o el promedio de una variable en un intervalo de tiempo. El operador introduce el identificador del punto a mostrar y el sistema busca los valores del punto en el intervalo de tiempo. El caso de uso termina cuando se muestra el mínimo valor, el máximo valor o el promedio de una variable en un intervalo de tiempo |
| **Complejidad** | Alta |
| **Prioridad** | Crítico |

| | |
|---|---|
| **Precondiciones** | Deben existir los puntos almacenados. |
| **Pos condiciones** | Se mostró el mínimo valor, el máximo valor o el promedio de una variable en un intervalo de tiempo. |
| **Flujo de eventos** | |
| **Flujo básico** | |
| **Actor** | **Sistema** |
| 1. El actor necesita mostrar el mínimo valor, el máximo valor o el promedio de una variable en un intervalo de tiempo. | 1.1 El sistema muestra las siguientes opciones:<br>a) Si decide mostrar el mínimo valor de una variable en un intervalo de tiempo ir a la sección “Mínimo”.<br>b) Si decide mostrar el máximo valor de una variable en un intervalo de tiempo ir a la sección “Máximo”.<br>c) Si decide mostrar el promedio de una variable en un intervalo de tiempo ir a la sección “Promedio”. |
| **Sección: “Mínimo”** | |
| **Actor** | **Sistema** |
| 1. Introduce el intervalo de tiempo. | 2. Verifica que el intervalo este correcto. En caso de que exista uno (Ver flujo alterno 1). |
| 3. Introduce el identificador del punto. | 4. Calcula el mínimo valor del punto en el intervalo de tiempo. |
| | 5. Verifica que no ocurra un error. En caso de error (Ver flujo alterno 2). |
| | 6. Muestra el mínimo valor del punto en el intervalo de tiempo. |
| | 7. Termina el caso de uso |
| **Sección: “Máximo”** | |

| **Actor** | **Sistema** |
|---|---|
| 1. Introduce el intervalo de tiempo. | 2. Verifica que el intervalo este correcto. En caso de que exista uno (Ver flujo alterno 1). |
| 3. Introduce el identificador del punto. | 4. Calcula el máximo valor del punto en el intervalo de tiempo. |
| | 5. Verifica que no ocurra un error. En caso de error (Ver flujo alterno 2). |
| | 6. Muestra el máximo valor del punto en el intervalo de tiempo. |
| | 7. Termina el caso de uso |
| **Sección: "Promedio"** | |
| **Actor** | **Sistema** |
| 1. Introduce el intervalo de tiempo. | 2. Verifica que el intervalo este correcto. En caso de que exista uno (Ver flujo alterno 1). |
| 3. Introduce el identificador del punto. | 4. Calcula el promedio del punto en el intervalo de tiempo. |
| | 5. Verifica que no ocurra un error. En caso de error (Ver flujo alterno 2). |
| | 6. Muestra el promedio del punto en el intervalo de tiempo. |
| | 7. Termina el caso de uso |
| **Flujos alternos** | |
| **Nº1. Ocurre un error al introducir el intervalo de tiempo.** | |
| **Actor** | **Sistema** |
| | 1. Muestra un mensaje de error "Debe introducir correctamente el intervalo de tiempo". |
| **Nº2. Ocurre un error al consultar los puntos almacenados.** | |
| **Actor** | **Sistema** |
| | 1. Muestra un mensaje de error "No existe la tabla". |

| | | |
|---|---|---|
| **Relaciones** | **CU Incluidos** | No tiene. |
| | **CU Extendidos** | No tiene. |
| **Requisitos funcionales** | RF 6, RF 7, RF 8 | |

**Tabla 10: Descripción del caso de uso "Mostrar agregados".**

## 2.5 Consideraciones parciales del capítulo.

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

# CAPÍTULO 3: DISEÑO, IMPLEMENTACIÓN Y PRUEBAS.

## 3.1 Introducción.

En este capítulo se procede a la realización del diseño e implementación del sistema. En él se exponen los diagramas de clases de diseño separados por paquetes, con la descripción de las clases que conforman a cada uno. Se presenta además el diagrama de componentes del sistema y por último se procede a la realización de pruebas al sistema, para validar su correcto funcionamiento.

## 3.2 Diseño del sistema.

Como se expuso en el capítulo anterior el enfoque principal del sistema es la gestión de datos históricos. En su diseño el sistema se divide en un grupo de paquetes que conforman su arquitectura, cada uno contiene un grupo de clases que se relacionan entre sí, de acuerdo con las funciones que realizan. Para el diseño de este sistema arquitectónico se utilizó el patrón n-capa como se muestra en la siguiente imagen.

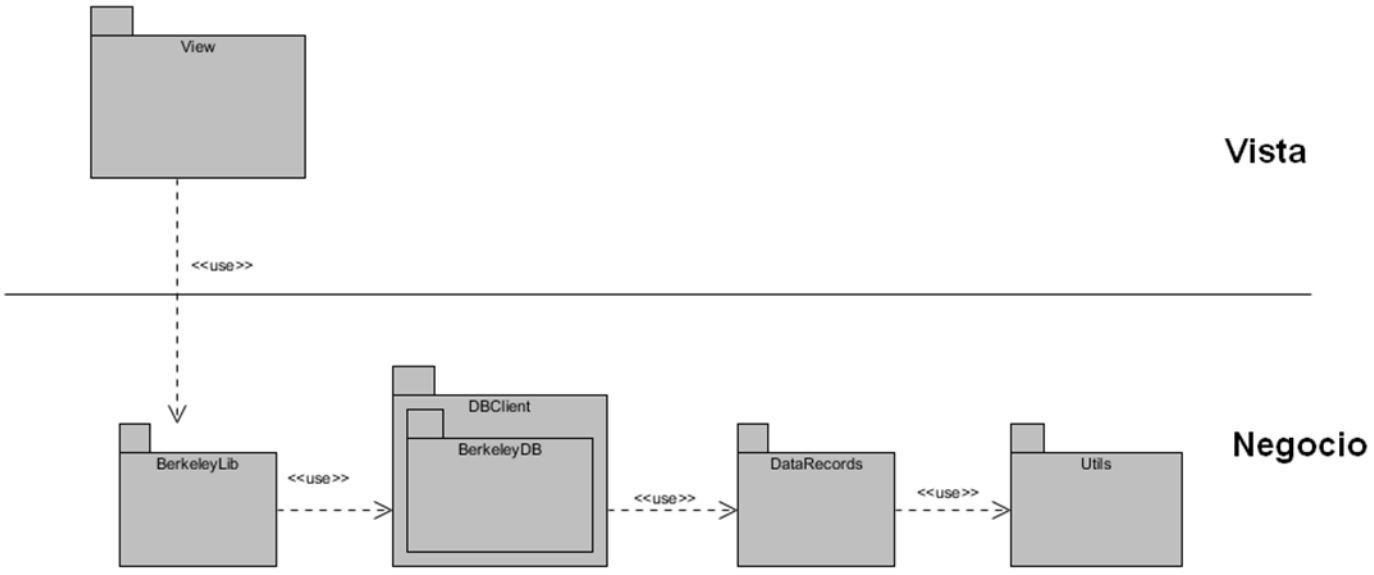



**Figura 6: Arquitectura en capas del sistema.**

En la capa de negocio se encapsulan los paquetes que contienen la funcionalidad del sistema, como son:

- ✓ “DBClient”: Este paquete contiene todas las clases relacionadas con la gestión de base de datos dentro del sistema.
- ✓ Subpaquete “BerkeleyDB”: Este subpaquete perteneciente al paquete DBClient, conforma la familia de objetos concretos para el SGBD embebido BerkeleyDB. El conjunto de clases que posee constituye la implementación de las interfaces definidas en el paquete DBClient.
- ✓ “DataRecords”: Este paquete contiene la estructura definida para el almacenamiento en la base de datos de las variables que controla el sistema.
- ✓ “Utils”: Este paquete contiene clases y métodos que poseen funcionalidades reutilizables por el paquete DataRecords.
- ✓ “BerkeleyLib”: Este paquete contiene la biblioteca desarrollada.

En la capa vista se encuentra el paquete View que contiene todas las interfaces gráficas con que el usuario interactúa. Permitiendo abstraer al usuario de la lógica del negocio.

### 3.2.1 Paquete View.

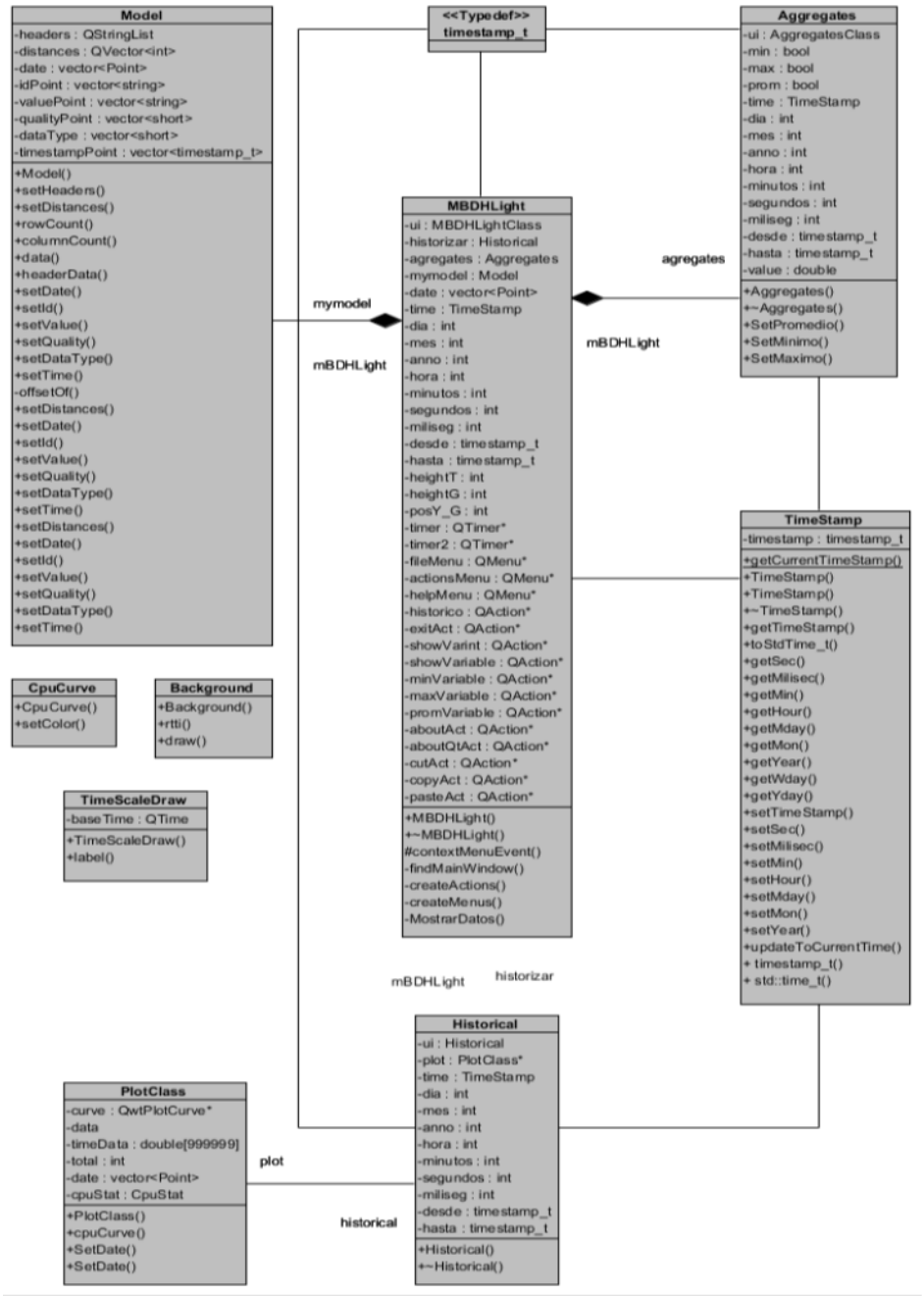



**Figura 7: Diagrama de clases del paquete View.**

- ✓ **Descripción de clases del paquete View.**

| Nombre de la clase | Descripción |
|---|---|
| Model | Esta clase representa el modelo del componente TableView, es donde se van a mostrar los datos almacenados. |
| MBDHLight | Clase principal del paquete. Esta clase implementa la interfaz de usuario principal, con la que el operador va a interactuar. |
| Aggregates | Esta clase implementa la interfaz de usuario donde se van a mostrar los agregados. |
| TimeStamp | Clase para definir el tipo de dato timestamp_t. Es la encargada de proporcionar un conjunto de funcionalidades para interactuar con el tiempo de las variables. |
| Historical | Esta clase implementa la interfaz de usuario donde se va a mostrar la tendencia de una variable en el tiempo. |
| PlotClass | Clase encargada de controlar todo el gráfico. |
| TimeScaleDraw | Clase encargada de graficar la escala de tiempo. |
| CpuCurve | Esta clase define el tipo de curva que se va a graficar con sus propiedades. |
| Background | Clase encargada del color de fondo de la gráfica. |

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

### 3.2.2 Paquete DBClient.

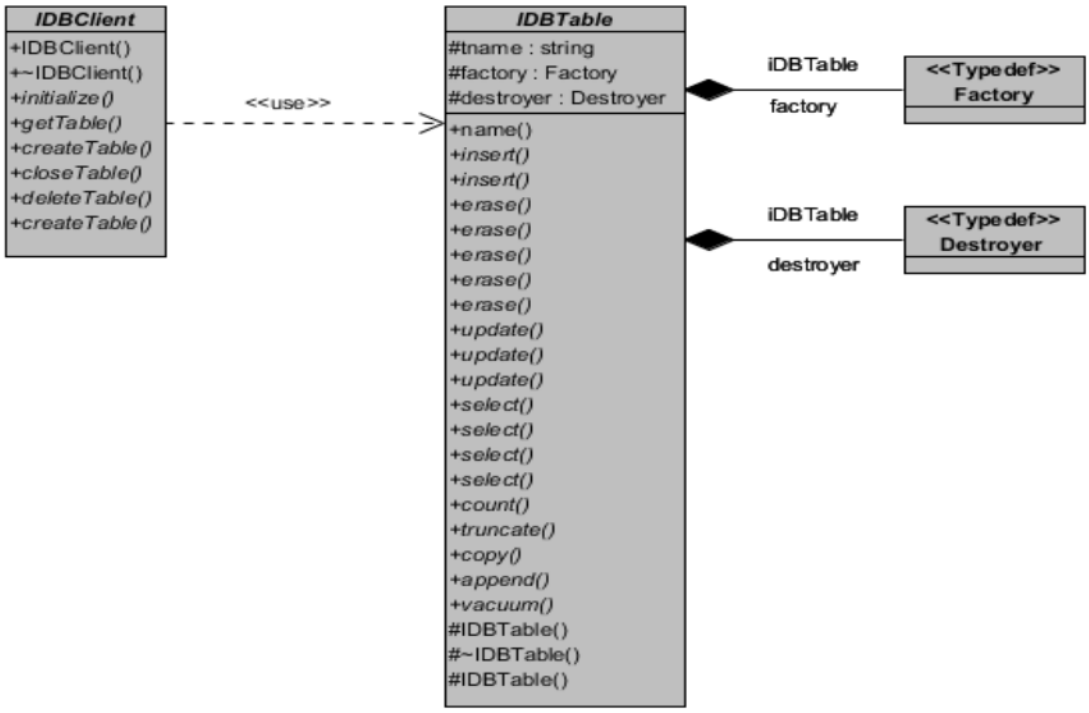



**Figura 8: Diagrama de clases del paquete DBClient.**

- ✓ **Descripción de clases del paquete DBClient.**

| Nombre de la clase | Descripción |
|---|---|
| IDBTable | Esta clase representa el concepto de tabla de base de datos. Expone un conjunto de funcionalidades que permiten interactuar con los datos almacenados. |
| IDBClient | Esta clase representa la interfaz a implementar por clases controladoras de base de datos. Expone un grupo de funcionalidades para la gestión de tablas de base de datos, es decir, instancias concretas de IDBTable. |

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

### 3.2.3 Subpaquete BerkeleyDB.

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

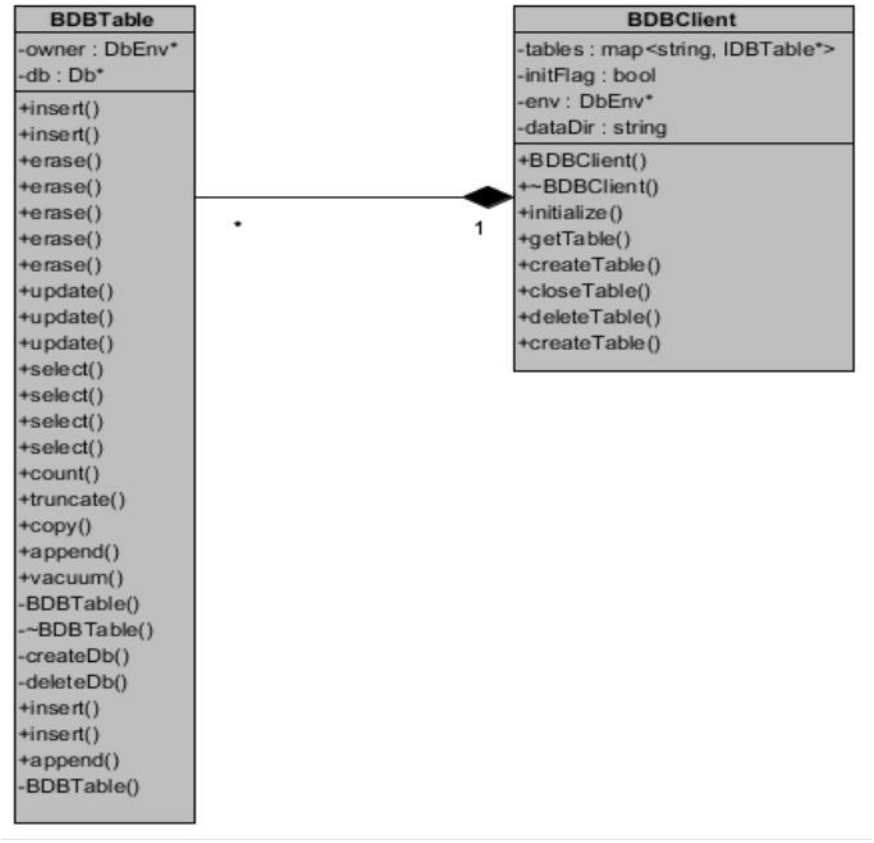



**Figura 9: Diagrama de clases del subpaquete BerkeleyDB.**

### 3.2.4 Paquete DataRecords.

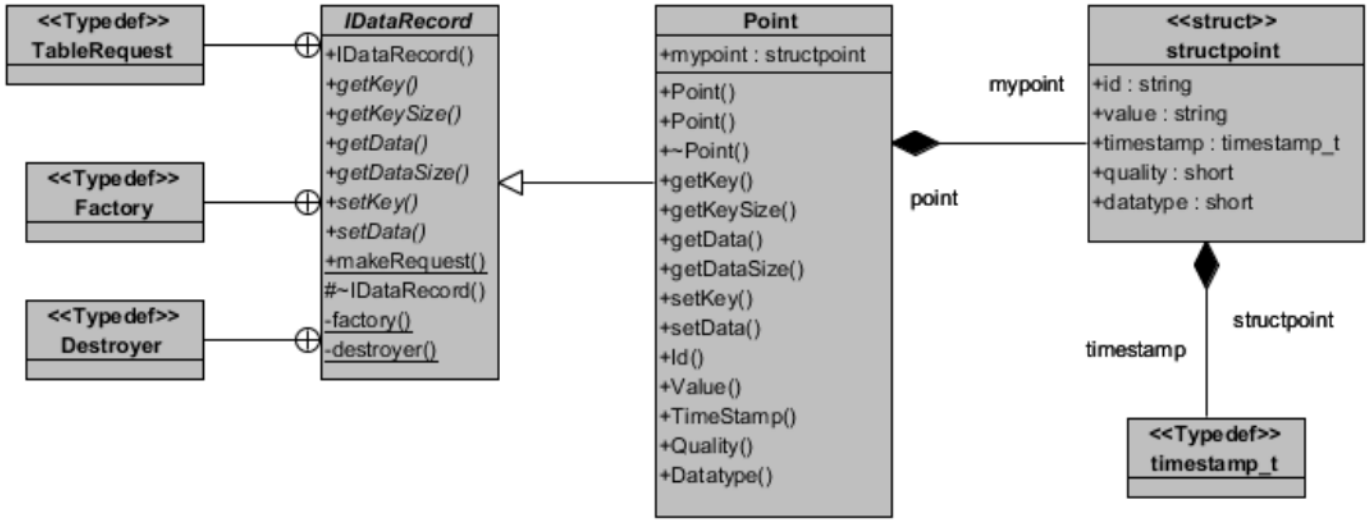



**Figura 10: Diagrama de clases del subpaquete DataRecords.**

- ✓ **Descripción de clases del paquete DataRecords.**

| Nombre de la clase | Descripción |
|---|---|
| IDataRecord | Las clases concretas que implementen esta interfaz constituyen la estructura definida para el almacenamiento en la base de datos de las variables que controla el sistema. |
| Point | Estructura definida para el almacenamiento en la base de datos de las variables que controla el sistema. |

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

## 3.3 Implementación del sistema.

### 3.3.1 Estilo de código.

**Nombres.**

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

**Manejo de errores.**

- ✓ Se pueden manejar los errores mediante mecanismos de excepciones o mediante valores de retorno, aunque esto debe ser uniforme dentro de un mismo objeto.
- ✓ Es buena práctica emplear herramientas para identificar errores en la codificación en caliente.

**Documentación y comentarios.**

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

**Codificación.**

- ✓ Se establece un tamaño de identación estándar de cuatro espacios, sin tabulaciones. Alinear secciones del código.
- ✓ Alinear verticalmente llaves de apertura y cierre.
- ✓ Usar espacios antes y después de los operadores que el lenguaje de programación permita.
- ✓ Emplear líneas en blanco para organizar el código, permitiendo crear párrafos de código para una mejor lectura.
- ✓ Evitar colocar más de una sentencia por línea.
- ✓ Emplear constantes en sustitución de números o cadenas de caracteres literales.
- ✓ Minimizar el alcance de las variables para evitar confusión y facilitar el mantenimiento.
- ✓ Emplear cada variable y rutina solo para un propósito.

- ✓ Evitar el uso de variables públicas, sustituirlas por variables privadas y métodos que provean el valor de tal variable, para mantener el encapsulamiento.
- ✓ Minimizar el uso de conversiones de tipo forzadas (castings), cuando se requiera su uso, debe ser comentada la justificación.
- ✓ Emplear select-case o switch en sustitución de if anidados sobre las mismas 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*, y si se conoce el número exacto de ciclos usar for.
- ✓ Inicializar todas las variables.
- ✓ Emplear líneas en blanco para separar pasos lógicos (declaraciones, lazos, etc.).
- ✓ Siempre asignar NULL a los apuntadores luego de ser destruidos (solo aplica para C).
- ✓ Evitar prácticas que incrementan explosivamente la complejidad, como lo son: objetos y variables globales y saltos tipo *goto*.

### 3.3.2 Diagrama de componentes.

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

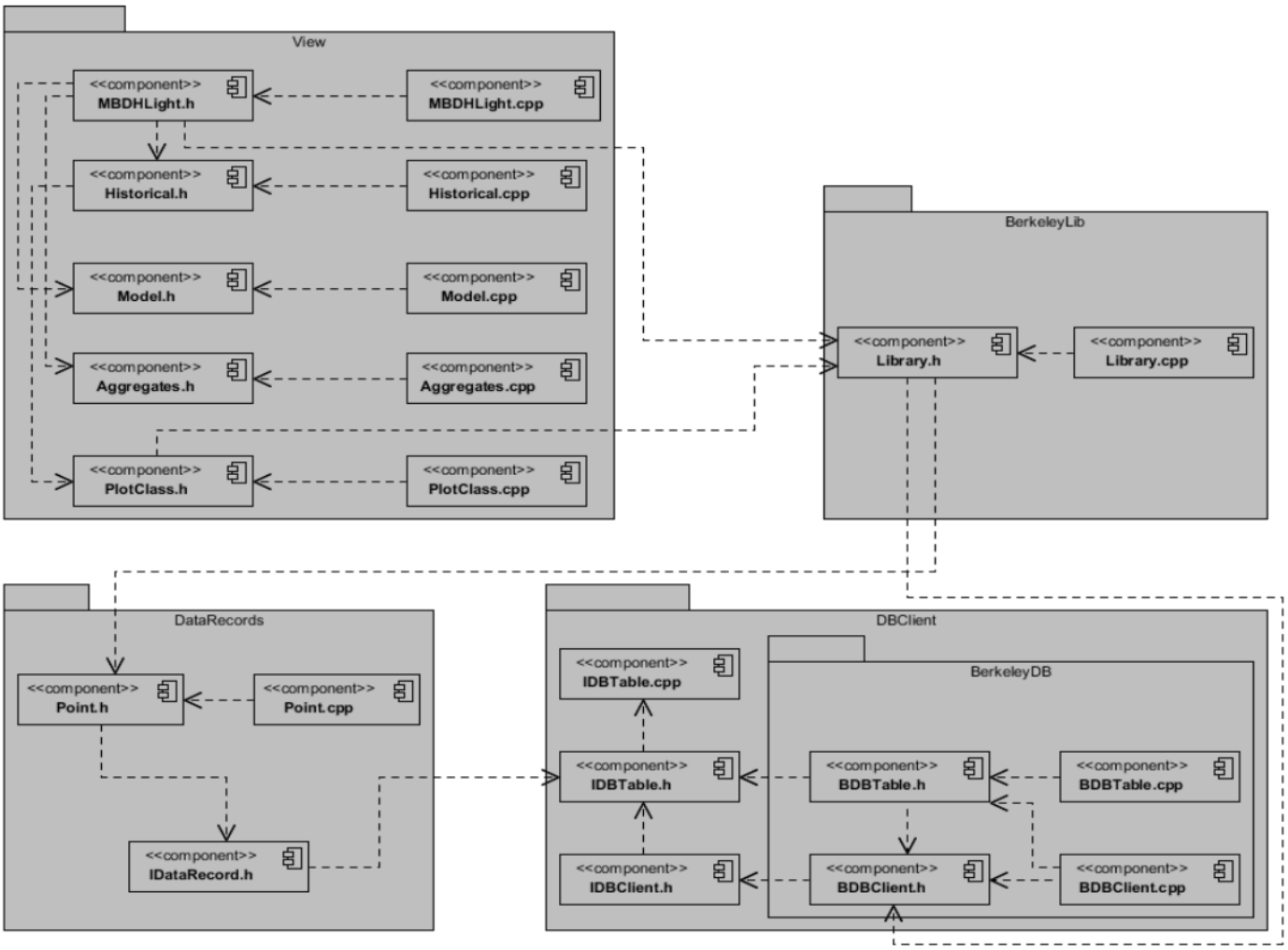



**Figura 11: Diagrama de componentes del sistema.**

## 3.4 Realización de pruebas.

Las pruebas de software son los procesos que permiten verificar y revelar la calidad de un producto de software. Donde las mismas son utilizadas para identificar posibles fallos de implementación, calidad o usabilidad de un software. Los dos métodos de prueba más utilizados son el método de prueba de caja blanca y el método de prueba de caja negra. La diferencia entre ambos consiste en que con el método de caja blanca es necesario conocer el código y estar bastante relacionado con él para saber exactamente cuál es la lógica interna de lo que se va a probar, sin embargo, en el método de caja negra solo basta con

conocer las posibles entradas y salidas del programa. A continuación se realizan una serie de pruebas al sistema basadas en el método de caja negra.

Para la realización de pruebas al sistema se definen unas variables aleatorias. Con las cuales se decide probar las funcionalidades generales de la biblioteca desarrollada. A continuación se presentan los casos de pruebas para dichas variables.

| **Caso de prueba 1** | | |
|---|---|---|
| **Acción del usuario** | **Respuesta esperada** | **Respuesta obtenida** |
| El sistema define el nombre de la base de datos y la ruta donde va a estar almacenada. Luego se ejecuta el sistema por consola pasando las variables aleatorias. | El sistema deberá almacenar de forma eficiente las variables en la base de datos definida anteriormente. | El sistema almacenó de forma eficiente las variables en la base de datos.<br><br>**Anexo 1** |
| **Caso de prueba 2** | | |
| **Acción del usuario** | **Respuesta esperada** | **Respuesta obtenida** |
| Con el sistema en ejecución, este decide borrar las tuplas sobrantes. | El sistema deberá borrar las tuplas sobrantes de la base de datos. | El sistema borró las tuplas sobrantes de la base de datos.<br><br>**Anexo 2** |
| **Caso de prueba 3** | | |
| **Acción del usuario** | **Respuesta esperada** | **Respuesta obtenida** |
| Con el sistema en ejecución, este decide borrar las tuplas que se encuentran almacenadas más tiempo del deseado. | El sistema deberá borrar las tuplas almacenadas más tiempo del deseado de la base de datos. | El sistema borró las tuplas almacenadas más tiempo del deseado de la base de datos.<br><br>**Anexo 3** |
| **Caso de prueba 4** | | |
| **Acción del usuario** | **Respuesta esperada** | **Respuesta obtenida** |
| Con el sistema en ejecución, se devuelven las variables | El sistema deberá devolver las variables almacenadas en la | Se devolvieron las variables almacenadas en la base de |

| | | |
|---|---|---|
| almacenadas en la base de datos. | base de datos. | datos.<br><br>**Anexo 4** |
| **Caso de prueba 5** | | |
| **Acción del usuario** | **Respuesta esperada** | **Respuesta obtenida** |
| Con el sistema en ejecución, se devuelven las variables almacenadas en un intervalo de tiempo. | En sistema deberá devolver las variables almacenadas en un intervalo de tiempo. | Se devolvieron las variables almacenadas en un intervalo de tiempo.<br><br>**Anexo 5** |
| **Caso de prueba 6** | | |
| **Acción del usuario** | **Respuesta esperada** | **Respuesta obtenida** |
| Con el sistema en ejecución, se devuelve el mínimo valor de una variable en un intervalo de tiempo. | El sistema deberá devolver el mínimo valor de una variable en un intervalo de tiempo. | Se devolvió el mínimo valor de una variable en un intervalo de tiempo.<br><br>**Anexo 6** |
| **Caso de prueba 7** | | |
| **Acción del usuario** | **Respuesta esperada** | **Respuesta obtenida** |
| Con el sistema en ejecución, se devuelve el máximo valor de una variable en un intervalo de tiempo. | El sistema deberá devolver el máximo valor de una variable en un intervalo de tiempo. | Se devolvió el máximo valor de una variable en un intervalo de tiempo.<br><br>**Anexo 7** |
| **Caso de prueba 8** | | |
| **Acción del usuario** | **Respuesta esperada** | **Respuesta obtenida** |
| Con el sistema en ejecución, se devuelve el valor promedio de una variable en un intervalo de | El sistema deberá devolver el valor promedio de una variable en un intervalo de tiempo. | Se devolvió el valor promedio de una variable en un intervalo de tiempo. |

| | | |
|---|---|---|
| tiempo. | | **Anexo 8** |

**Tabla 14: Pruebas realizadas al sistema.**

La segunda parte de las pruebas realizadas fueron a la aplicación visual, utilizado el método de caja negra. Donde se decide probar las principales funcionalidades con que cuenta la aplicación visual. A continuación se representan los casos de pruebas.

| **Caso de prueba 9** | | |
|---|---|---|
| **Acción del usuario** | **Respuesta esperada** | **Respuesta obtenida** |
| Con la aplicación en ejecución, se decide mostrar las variables almacenadas. | El sistema deberá mostrar las variables almacenadas en una tabla. | Se mostraron las variables almacenadas en una tabla.<br><br>**Anexo 9** |
| **Caso de prueba 10** | | |
| **Acción del usuario** | **Respuesta esperada** | **Respuesta obtenida** |
| Con la aplicación en ejecución, se decide mostrar la tendencia de una variable en un intervalo de tiempo. El usuario introduce el identificador de la variable, así como el intervalo de tiempo y presiona el botón graficar. | El sistema deberá mostrar la tendencia de la variable en el intervalo de tiempo. | El sistema mostró la tendencia de la variable en el intervalo de tiempo.<br><br>**Anexo 10** |
| **Caso de prueba 11** | | |
| **Acción del usuario** | **Respuesta esperada** | **Respuesta obtenida** |
| Con la aplicación en ejecución, se decide mostrar el valor promedio de una variable en un intervalo de tiempo. El usuario introduce el identificador de la | El sistema deberá mostrar el valor promedio de la variable en el intervalo de tiempo. | El sistema mostró el valor promedio de la variable en el intervalo de tiempo.<br><br>**Anexo 11** |

| | | |
|---|---|---|
| variable, así como el intervalo de tiempo y presiona el botón mostrar promedio. | | |

**Tabla 15: Pruebas realizadas a la aplicación.**

## 3.5 Consideraciones parciales.

En el capítulo se presentó el diseño de las clases y paquetes que estructuran al módulo de gestión y archivo de datos. Fueron implementadas todas las clases del diseño, teniendo como resultado un producto que cumple con los requisitos funcionales descritos en el capítulo anterior. Como último paso se realizaron un conjunto de pruebas al sistema que validaron su correcto desempeño.

# CONCLUSIONES

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

- ✓ Se desarrolló un módulo de gestión y archivo de datos para el SCADA Ligero que permite el almacenamiento de datos de forma eficiente. Además de una aplicación visual independiente al sistema para visualizar dichos datos almacenados.
- ✓ Se utilizó Berkeley DB como sistema gestor de base de datos empotrado.
- ✓ Se realizaron una serie de pruebas al sistema que arrojaron resultados satisfactorios.
- ✓ Se obtuvo un producto fácil de desplegar y operar, con mínimos recursos de memoria y procesamiento, pensado para escenarios que no involucren la manipulación de grandes volúmenes de datos y requieran de un rápido acceso a los mismos.

# RECOMENDACIONES

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

- ✓ Incorporar a la biblioteca más funcionalidades para gestionar alarmas, eventos y bitácoras.
- ✓ Agregar a la aplicación visual nuevas funcionalidades para que permita la visualización de alarmas, eventos y bitácoras.

# REFERENCIAS BIBLIOGRÁFICAS.

1. **UCI.** Portal de la Universidad de las Ciencias Informáticas. [Online] [Citado: 12 de Octubre, 2011.] http://www.uci.cu/mision.

2. **Sánchez Briceño, Gabriel; Custodio Ruiz, Ángel y Zerpa, Héctor.** *Servidor para un sistema de supervisión y control de procesos industriales bajo software libre*. Puerto Ordaz. Universidad de Ciencia y Tecnología. 2010.

3. **Mendiburu Díaz, Henry.** *Sistemas SCADA.* s.n.

4. **Chacon, David, Dijort, Oscar y Castillo, Jacinto.** *Supervisión y Control de Procesos.* 2001-2002.

5. **Jiménez, Verónica Consuelo.** *Estudio de factibilidad de un sistema de comunicación integrado al sistema microonda existente, que soporte aplicaciones de sistemas Scada para el Poliducto Esmeraldas Quito de Petrocomercial regional norte.* s.l. : QUITO/ EPN, 2008.

6. Scadas. [En línea] 02 de 03 de 2006. [Citado el: 17 de 04 de 2012.] http://www.automatas.org/redes/scadas.htm.

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

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

9. Foundation, OPC. *OPC Historical Data Access Specification*. 2003.

10. *SQLite, el motor de base de datos ágil y robusto.* **Maldonado, Daniel Martin.** 2008º.

11. **Soulie, Juan.** *cplusplus.com.* [En línea] cplusplus.com, 09 29, 2009. [Citado: 21 de 04 de 2012.] http://cplusplus.com/info/description/.

# BILIOGRAFÍA.

1. Oracle Berkeley DB. [En línea] Oracle. [Citado el: 28 de marzo de 2011.] http://www.oracle.com/database/berkeley-db/db/index.html.

2. Oracle Berkeley DB 11g. [En línea] Oracle. [Citado el: 28 de marzo de 2011.] http://www.oracle.com/technology/products/berkeley-db/index.html.

3. Oracle Berkeley DB Products. [En línea] Oracle. [Citado el: 28 de marzo de 2011.] http://www.oracle.com/database/berkeley-db/index.html.

4. Sitio oficial de SQLite. [En línea] [Citado el: 28 de marzo de 2011.] http://www.sqlite.org/.

5. *Los sistemas gestores de bases de datos y la red.* [En línea] La tecla de escape. [Citado el: 17 de abril de 2012.] http://latecladeescape.com/w0/basico/los-sistemas-gestores-de-bases-de-datos-y-la-red.html.

6. *solidDB product family.* [En línea] IBM. [Citado el: 10 de abril de 2012.]

http://www-01.ibm.com/software/data/soliddb/.

7. **Camps Paré, Rafael, y otros.** *Base de Datos.* Catalunya: s.n., 2005. 12. **Chavarría Meza, Luis Eduardo.** *SCADA SYSTEM´S & TELEMETRY.* México: s.n., 2007. Módulo de Gestión y Archivo de Datos para el sistema SCADA.

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

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

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

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

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

# GLOSARIO DE TÉRMINOS.

**Biblioteca:** Es un conjunto de subprogramas para desarrollar software.

**LAN:** Red de área local (por sus siglas en inglés *Local Area Network*). Define la conexión física y lógica de ordenadores en un entorno generalmente de oficina.

**SCADA:** Supervisión, control y adquisición de datos (por sus siglas en inglés ***S****upervisory* ***C****ontrol* ***a****nd* ***D****ata* ***A****cquisition*).

**HDAIS:** Acceso a datos históricos de sistemas industriales (por sus siglas en inglés *Historical Data Access for Industrial Systems*).

**OPC HDA**: (Historical Data Access, Acceso a Datos Históricos) Especificación OPC dedicado al acceso a datos históricos exclusivamente.

**UML:** Lenguaje unificado de modelado (UML, por sus siglas en inglés, *Unified Modeling Language*). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

**RUP:** (*Rational Unified Process* en inglés) Es una infraestructura flexible de desarrollo de software que proporciona prácticas recomendadas probadas y una arquitectura configurable.

**CASE:** (*Computer Aided Software Engineering*, ingeniería de software asistida por ordenador) Son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software.

**QT:** Es un framework multiplataforma utilizado principalmente para desarrollar interfaces gráficas de usuario.

**IDE:** Entorno de desarrollo integrado (acrónimo en inglés de *Integrated Development Environment*), es un programa informático compuesto por un conjunto de herramientas de programación.

**PLC:** Es un hardware industrial, que se utiliza para la obtención de datos.