



Universidad de las Ciencias Informáticas

Facultad 5

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas

Mecanismo de caché para la biblioteca de acceso a datos históricos del SCADA Galba

Autor: Luis Felipe Domínguez Vega

Tutores: Ing. Yusnier Manuel Sosa Vázquez
Ing. Dairon Domínguez Vega

La Habana, junio de 2015

Declaración de autoría

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

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

Luis Felipe Domínguez Vega

Jefe de línea de Históricos
Ing. Yusnier Manuel Sosa Vázquez

Especialista C en Ciencias Informáticas
Ing. Dairon Domínguez Vega

Datos de contacto

Autor:

Luis Felipe Domínguez Vega
Universidad de las Ciencias Informáticas, La Habana, Cuba.
Correo Electrónico: lfdominguez@estudiantes.uci.cu

Tutores:

Ing. Yusnier Manuel Sosa Vázquez
Universidad de las Ciencias Informáticas, La Habana, Cuba.
Correo Electrónico: ymvazquez@uci.cu

Ing. Dairon Domínguez Vega
Desoft, Matanzas, Cuba.
Correo Electrónico: ddvega@nauta.cu

AGRADECIMIENTOS

Agradezco a todas las personas que hicieron posible la realización de este proyecto. A mis padres y mi hermano por trazar el camino a seguir para ser un buen profesional, incluyendo a Mayling claro. A mi novia Eli, que tuvo que aguantar “el modo tesis”. A Albert y Celia por su apoyo incondicional. A Isabel por haberme apoyado tanto en la realización de esta tesis. A mi niña “Envy” por estar presente en todo momento. A las personas que se convirtieron en mis más allegados amigos, Lorgio, los que vienen conmigo desde la vocacional, Ale, Poty, Enmanuel, David, Raidel, Rydel; a Luis Miguel, Leiser, Yuriasky, Nelson, Rioger, etc... A Lara, Koko, Kiki, Cuba y a toda mi aula en general que fuimos y seremos la mejor brigada de la facultad 5. A mis tutores, de los cuales cada día aprendía algo nuevo. Hubo personas que no mencioné, no dejan de tener mi agradecimiento, solo es cuestión que se olvida alguien siempre... pero que tengan presente que siempre me recordaré de todos.

DEDICATORIA

A Mima, por aguantar mis malcriadeces.

A Tito, por ser el científico que me ha enseñado las cosas que sé.

A Mano, por ser la competencia que me impulsó a superarme y mi guía a seguir.

RESUMEN

En muchos sistemas SCADA, como el Galba, se almacena toda la información histórica en una base de datos, a lo largo del tiempo que se encuentra funcionando. Para acceder a dicha información se hace uso de una biblioteca especial, encargada de su obtención directamente de la base de datos de históricos, entregándolo a la aplicación que lo solicite. Cuando se realizan pedidos que involucran grandes cantidades de datos la demora del proceso de consulta aumenta, impidiendo el trabajo fluido del operador. Esta tesis presenta una propuesta para disminuir el tiempo de respuesta, cuando los pedidos de datos históricos contienen rangos de tiempo solapados. Se hace uso de un sistema de caché que almacena temporalmente, tanto en RAM como en HDD, cada elemento obtenido desde la base de datos. Todas las bibliotecas utilizadas son *software libre*, brindando la posibilidad de su inclusión en *software* privado gracias a la licencia que utilizan. Se emplea la variante UCI de la metodología de desarrollo AUP, describiendo el diseño e implementación de la solución.

Palabras Claves: *caché, cliente-servidor, data-shipping, c++, boost, msgpack, aup, histórico*

ÍNDICE GENERAL

	Página
Índice de figuras	VII
Índice de tablas	VIII
Introducción	1
1. Fundamentos teóricos	5
1.1. Introducción	5
1.2. Mecanismo de Caché	6
1.2.1. Términos comunes	7
1.3. Caché en arquitectura cliente-servidor	9
1.4. Almacenamiento de la caché	10
1.4.1. Serialización de datos en el HDD	11
1.5. Políticas de reemplazo en la caché	15
1.5.1. Algoritmo Bélády's	15
1.5.2. Algoritmo LRU	16
1.5.3. Algoritmo MRU	16
1.5.4. Algoritmo LFU	17
1.5.5. Otros algoritmos	17
1.6. Metodología de desarrollo	18
1.6.1. AUP	18
1.7. Herramientas y tecnologías	20
1.7.1. Lenguaje de modelado	20

1.7.2. Sistemas homólogos	20
1.7.3. Herramienta para el modelado de diagramas	21
1.7.4. Lenguaje de programación	21
1.7.5. IDE	22
1.7.6. Bibliotecas de desarrollo	23
1.7.7. Sistema de Gestión de base de datos	23
1.8. Consideraciones del capítulo	24
2. Propuesta de solución	25
2.1. Introducción	25
2.2. Descripción del proceso a automatizar	25
2.3. Modelo de dominio	26
2.3.1. Diagrama de Clases del Dominio	26
2.3.2. Descripción de las clases del modelo de dominio	26
2.4. Especificación de requisitos de <i>software</i>	27
2.4.1. Requisitos funcionales	27
2.4.2. Descripción de las Historias de Usuario	28
2.4.3. Requisitos no funcionales	35
2.5. Propuesta de solución	35
2.6. Arquitectura de la Biblioteca de Acceso a Datos de Históricos (libHDA, por sus siglas en inglés)	36
2.6.1. Estilo y patrones arquitectónicos	37
2.6.2. Patrones de diseño	37
2.7. Modelo de diseño	38
2.7.1. Diagramas de Clases del Diseño	38
2.7.2. Descripción de las Clases del Diseño	38
2.8. Modelo de Despliegue	44
2.9. Consideraciones del capítulo	45
3. Implementación y pruebas	47
3.1. Introducción	47
3.1.1. Selección de biblioteca de serializado	47
3.2. Modelo de implementación	49
3.2.1. Diagramas de componentes	49
3.3. Modelo de Pruebas	49
3.3.1. Diseño de casos de prueba	51
3.3.2. Framework para pruebas “ <i>Google C++ Testing Framework</i> ”	54
3.3.3. Validación de la propuesta a través de la eficiencia y eficacia	54

3.4. Consideraciones del capítulo	58
Conclusiones	59
Recomendaciones	60
Acrónimos	61
Referencias bibliográficas	64

ÍNDICE DE FIGURAS

	Página
1.1. Llamadas a funciones, analizado con Callgrind, visualizado con KCachegrind	5
1.2. Llamadas a funciones, analizado con Intel® VTune™ Amplifier XE 2013	6
1.3. Ejemplo gráfico de solapamiento en los pedidos de datos a la base de datos	6
1.4. Caché <i>Hit</i>	7
1.5. Caché <i>Miss</i>	8
1.6. Relación entre la velocidad y la capacidad de almacenamiento entre RAM y HDD .	10
1.7. Algoritmo Bélády's	16
1.8. Algoritmo LRU	16
1.9. Algoritmo MRU	16
1.10. Algoritmo LFU	17
2.1. Diagrama de clases del dominio	26
2.2. Diagrama de clases del diseño	45
2.3. Diagrama de despliegue	45
3.1. Diagrama de componentes	50
3.2. Salida acotada de las pruebas con el framework Google Test	55
3.3. Salida para la clase <code>AbstractQuery</code> del código cubierto por las pruebas	55

ÍNDICE DE TABLAS

	Página
2.1. RF1 Almacenamiento en RAM de los datos temporales.	28
2.2. RF1.1 Almacenamiento en RAM de puntos analógicos.	28
2.3. RF1.2 Almacenamiento en RAM de puntos digitales.	28
2.4. RF1.3 Almacenamiento en RAM de logs del sistema.	29
2.5. RF2 Almacenamiento en HDD de los datos temporales.	29
2.6. RF2.1 Almacenamiento en HDD de puntos analógicos.	30
2.7. RF2.2 Almacenamiento en HDD de puntos digitales.	30
2.8. RF2.3 Almacenamiento en HDD de logs del sistema.	31
2.9. RF3 Configurar espacio de almacenamiento en RAM.	31
2.10. RF4 Configurar espacio de almacenamiento en HDD.	32
2.11. RF5 Configurar la dirección lógica en el HDD donde serán almacenados los datos de la caché.	32
2.12. RF6 Seleccionar el algoritmo de reemplazo para datos de la caché.	32
2.13. RF7 Ordenar los datos devueltos por <i>timestamp</i>	33
2.14. RF8 Limpiar la caché.	33
2.15. RF9 Habilitar y deshabilitar el mecanismo de caché.	34
2.16. RF10 Consultar varios identificadores de un mismo tipo.	34
2.17. RF11 Recuperarse ante fallas de conexión a la base de datos.	34
2.18. Descripción de CacheSystem.	38
2.19. Descripción de CacheSystemPoint.	40
2.20. Descripción de CacheSystemLog.	41
2.21. Descripción de AbstractQuery<Type>.	42
2.22. Descripción de CacheQuery<Type>.	42

2.23.Descripción de DiskQueryLog.	43
2.24.Descripción de DiskQueryPoint.	44
3.1. Pruebas realizadas para determinar velocidad en milisegundos de la serialización. .	48
3.2. Pruebas realizadas para determinar el tamaño de los datos en megabytes de la serialización.	48
3.3. Casos de prueba.	51
3.4. Casos de prueba unitarias.	53
3.5. Prueba realizada sin resultados de puntos.	56
3.6. Prueba realizada para 197'957 puntos.	56
3.7. Prueba realizada para 395'906 puntos.	56
3.8. Prueba realizada para 494'740 puntos.	56
3.9. Prueba realizada para 538'280 puntos.	56
3.10.Prueba realizada para un 25 % de convergencia de datos	57
3.11.Prueba realizada para un 50 % de convergencia de datos	57
3.12.Prueba realizada para un 75 % de convergencia de datos	57
3.13.Prueba realizada para un 100 % de convergencia de datos	57

INTRODUCCIÓN

Los sistemas de Supervisión, Control y Adquisición de Datos (SCADA, por sus siglas en inglés), como *software* de monitoreo y control, son aplicaciones utilizadas para monitorizar los procesos industriales, brindando al supervisor u operador que los controla una idea general del estado, en todo momento, del despliegue¹ realizado. De ahí que se permita tomar acciones dependiendo de una situación determinada.

Con la creación de la UCI, nuestro país ha incrementado el desarrollo de proyectos de perfil industrial. Específicamente en la facultad 5 se encuentra el Centro de Informática Industrial (CEDIN), en el cual, el Departamento de Construcción de Componentes está inmerso en el desarrollo de un sistema para el monitoreo y control de los procesos petrolíferos de Venezuela, SCADA Guardián del ALBA (Galba). Alguno de los componentes que se desarrollan son:

Interfaz Hombre-Máquina (HMI, por sus siglas en inglés): Brinda un ambiente visual para que los operadores y diseñadores del sistema, configuren y monitoreen de manera automatizada todo el ambiente. En él se define un despliegue acorde al proceso industrial a monitorizar, permitiendo una mayor familiarización y ubicación espacial de los componentes.

Histórico: Almacena en una base de datos todo el estado de las variables del sistema SCADA a partir de su tiempo de recolección.

libHDA: Biblioteca encargada de devolver los datos almacenados en el servidor, a través de consultas realizadas por aplicaciones clientes, como es el caso de HMI.

La cantidad de datos generados depende del espacio asignado para el almacenamiento, el tiempo que haya transcurrido desde que se comenzó a almacenar la data y la configuración que se haya realizado sobre el tiempo entre cada muestra de datos². Cuando los operadores requieren de un trabajo de análisis de elementos históricos, realizan consultas indicando un intervalo de

¹Se refiere a la configuración personalizada para la propia empresa.

²Conocido como *Sample Rate* o Período de Muestreo.

tiempo, las cuales, por su naturaleza involucran grandes cantidades de datos (llegando incluso al orden de los *gigabytes*). La mayoría de las veces estos datos se asocian con su presentación a largo plazo, ya sea para mostrar su comportamiento en etapas de tiempo variables (desde los días hasta los años) o cuando se involucran en grandes cálculos relacionados con la producción. Estas características provocan que muchas veces, los tiempos de acceso de la libHDA a los datos solicitados no sean aceptables para el operador. Por lo que se define como **problema científico** de la investigación:

¿Cómo disminuir el tiempo de acceso a datos históricos consultados a través de la biblioteca de acceso a datos del SCADA Galba?

Siendo el **objeto de estudio** los mecanismos de caché en entornos de base de datos con arquitectura cliente-servidor y como **campo de acción** los mecanismos de caché en sistemas SCADA.

Se plantea como **objetivo general**, desarrollar un mecanismo de caché para la reutilización de los datos devueltos por las consultas de la Biblioteca de Acceso a Datos de Históricos. Teniendo como objetivos específicos:

- Analizar las metodologías, sistemas y herramientas existentes de reutilización de datos.
- Identificar los requerimientos del sistema.
- Definir la arquitectura del sistema a desarrollar.
- Elaborar una documentación como guía para futuros desarrollos sobre el sistema creado.

Originando como **posibles resultados**:

- Artefacto del análisis, diseño e implementación de las funcionalidades a incorporar dentro de la libHDA.
- Un mecanismo para la reutilización de los datos devueltos por el Servidor de Históricos en la libHDA.
- Documentación como guía para futuros desarrollos sobre el sistema creado.

El autor defiende la idea de que si se implanta el mecanismo de caché para la biblioteca de acceso a datos históricos del SCADA Galba, se logrará disminuir el tiempo de consulta a estos datos, se humaniza el trabajo de los operadores, aumenta la agilidad en las respuestas a las solicitudes realizadas y se logra ejecutar mayor cantidad de operaciones con economía de recursos.

Los resultados de la presente investigación contribuyen a la práctica, al implementar un mecanismo que, aprovechando las bondades de la caché, contribuye a la eficiencia en la utilización de los sistemas SCADA, pues disminuye el tiempo de consulta a los datos históricos que se almacenan en la base de datos, propiciando mejores resultados de trabajo y ahorro de tiempo.

La investigación utilizó métodos científicos, bajo la concepción dialéctica-materialista. Utilizando como métodos teóricos:

Histórico-Lógico: Para la determinación de los antecedentes en su devenir histórico, tendencias y regularidades del objeto de estudio y campo de acción, así como un estudio cronológico sobre mecanismos de caché que realizan tareas similares, de donde se obtuvieron conocimientos que fueron aplicados en la solución del problema planteado.

Análisis-Síntesis e Inductivo-Deductivo: Para la determinación de las generalidades y especificidades en el objeto de estudio y el campo de acción, así como la fundamentación teórica de la idea que se defiende. Todo el estudio realizado para desarrollar la solución se sustenta en éste método.

Modelación: Se utilizó con el objetivo de generar los artefactos necesarios y para diseñar la solución propuesta. Se muestra en todos los modelos esclarecedores de la solución al problema. Entre ellos se puede encontrar los diagramas de clases del diseño, diagrama de componentes, etc.

Se utilizaron como métodos empíricos:

Observación científica: Con el objetivo de obtener una información precisa y real del fenómeno que se estudia, en este caso, el proceso de obtención de los datos por parte de la biblioteca de acceso a datos al servidor de históricos. Permitió detectar las necesidades existentes y posibles mejoras a incorporar al *software*, con el propósito de extender sus funcionalidades y alcanzar un producto de mayor calidad.

Consulta de la información en todo tipo de base: Se consulta a través de libros, sitios web, en general, cualquier fuente de información; para la búsqueda de conceptos que esclarezcan todos los elementos a tener en cuenta para la solución del problema. Permitió la elaboración del marco teórico de la investigación, en sus aspectos conceptuales y metodológicos. Todas las referencias bibliográficas utilizadas son debidamente descritas.

Nivel estadístico: Mediante la estadística descriptiva, con el empleo de gráficos, tablas, porcentajes, se hace posible la organización y regularización de la información obtenida. Se ha utilizado para demostrar la eficiencia y eficacia lograda con la solución del problema, realizando pruebas, las cuales a través de gráficos se demuestran los resultados obtenidos, como son las de serializado y de aceptación.

En la investigación se proponen las tareas siguientes:

- Revisión bibliográfica para generar el marco teórico conceptual de la investigación, que represente las tendencias actuales en los mecanismos de reutilización de datos.
 - Análisis de los sistemas y herramientas existentes para la reutilización de datos.
- Selección de la metodología de desarrollo y las tecnologías a utilizar.
- Identificación y modelación de los requerimientos del sistema.
- Definición de la arquitectura del sistema a desarrollar y modelar la propuesta.
- Diseño de un modelo de clases que responda a las exigencias del sistema a desarrollar.
- Implementación del sistema modelado.
- Validación del sistema implementado.
- Creación de la documentación que recoja todo el proceso de implementación del sistema.

La memoria escrita de la investigación está estructurada en tres capítulos:

- **Capítulo 1:** Fundamentos teóricos. Sustentan los mecanismos para la reutilización de datos en la libHDA del SCADA Galba. En este capítulo se presenta una reseña bibliográfica donde se hace un análisis de la literatura y una presentación de la información recopilada que está estrechamente relacionada con el tema tratado. Se plantean las herramientas y metodologías utilizadas en la elaboración del presente trabajo de diploma.
- **Capítulo 2:** Propuesta de solución. Se describe el proceso a automatizar, el modelo de dominio, requisitos de *software*, incluyendo como se implementará y desplegará la solución.
- **Capítulo 3:** Implementación y pruebas. Se plantea como sería implementado el mecanismo de reutilización de datos en la libHDA, así como su eficiencia comparándose con la versión sin utilizar dicho mecanismo.

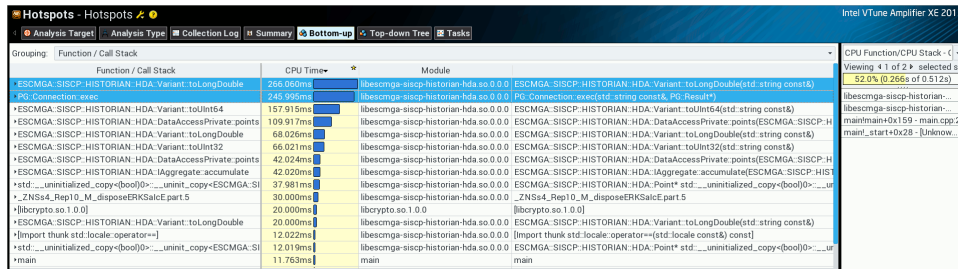


Figura 1.2. Llamadas a funciones, analizado con Intel® VTune™ Amplifier XE 2013

incidir en que se realicen consultas de datos redundantes, gráficamente se observa un ejemplo en la Figura 1.3



Figura 1.3. Ejemplo gráfico de solapamiento en los pedidos de datos a la base de datos

1.2. Mecanismo de Caché

En el ámbito informático, es un mecanismo intermedio utilizado para almacenar temporalmente datos que son accedidos con elevada frecuencia. El proceso consiste en guardar en un determinado lugar de almacenamiento dichos datos que, al solicitarse, incurren en un elevado coste computacional, ya sea por el transporte o por la necesidad de volver a realizar cálculos³ complejos.

Principalmente se evidencia en las arquitecturas cliente-servidor. La comunicación con este último, además del procesamiento de los datos a retornar, incurre en una demora para los pedidos del cliente; por este motivo se mantiene una copia local, llamada caché, de los datos más accedidos por el cliente. De esta manera se evita el costo de generación más transporte de los datos.

Otro de los ámbitos donde se encuentra el uso de una caché es en los microprocesadores, en la cual es una pequeña memoria de alta velocidad, generalmente es más rápida que la Memoria de Acceso Aleatorio (RAM, por sus siglas en inglés), se nombra RAM Estática (SRAM, por sus siglas en inglés). Esta última contiene las piezas más accedidas recientemente de la RAM principal [Intel(2010)], de esta caché no se abarcará en el presente documento.

³no solo operaciones matemáticas, sino toda aquella recopilación de información y transformación hacia otro tipo.

Los sistemas de caché, en arquitecturas cliente-servidor, pueden ubicarse tanto del lado del cliente como del lado del servidor, dependiendo de la implementación y de la situación problemática que se presente. El proceso de funcionamiento se basa en que, la primera vez que los datos son accedidos, se realiza una copia donde pueden ser consultados de forma más rápida. De esta manera los datos son accedidos con menos coste de tiempo, evitando los costes de la generación y transporte.

1.2.1. Términos comunes

Cuando se habla de mecanismos de caché se definen algunos términos propios de su trabajo, los cuales se explicarán en las secciones siguientes.

Hit

Cuando la caché contiene la información pedida, se indica que la transacción es un *Hit*. Se observa un ejemplo (Figura 1.4).

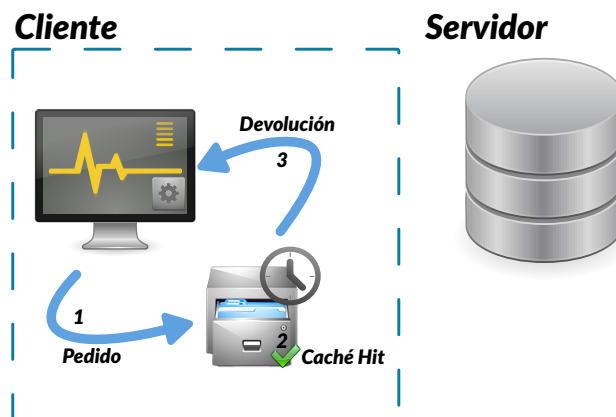


Figura 1.4. Caché *Hit*

Este proceso se puede describir (refiriéndonos a Figura 1.4) de la manera siguiente:

1. Se piden los datos a la caché.
2. La caché verifica que tiene los datos (***Hit***).
3. Devuelve los datos sin necesidad de obtenerlos del servidor.

Miss

Cuando la caché no contiene la información pedida, se indica que la transacción es un *Miss*. Un ejemplo en la Figura 1.5.

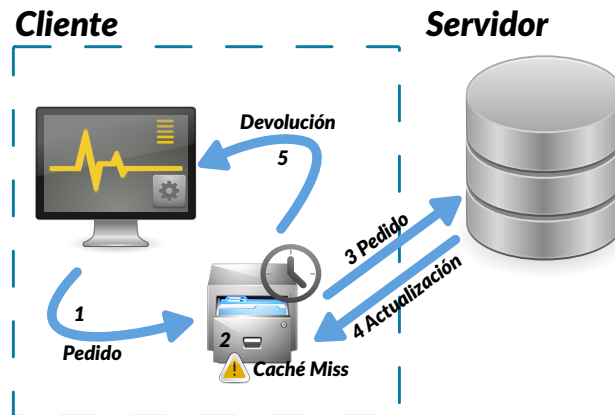


Figura 1.5. Caché Miss

Este proceso se puede describir (refiriéndonos a la Figura 1.5) de la manera siguiente:

1. Se piden los datos a la caché.
2. La caché verifica que no cuenta con estos datos (**Miss**).
3. Se realiza un pedido a la base de datos.
4. La caché actualiza sus registros con los nuevos datos.
5. Se devuelven los datos

Se debe tener en cuenta que para el *Miss* el costo es mayor, con respecto al tiempo necesario para obtener el dato requerido por el cliente. Esto es debido al paso intermedio que implica actualizar la caché, pero si se accede con frecuencia a los datos en un futuro, disminuye la suma total del consumo de tiempo.

Consistencia de la caché

Como la caché es una copia exacta de una porción pequeña de la memoria principal del sistema, es importante que siempre refleje los datos que hay realmente en la memoria principal. Algunos términos son utilizados para describir el proceso de mantener la consistencia en la caché, como:

Snoop (Monitorizar): Se llama *Snoop* cuando la caché observa las líneas de las transacciones, esto permite ver si los datos que se están solicitando se encuentran dentro de ella [Intel(2010)].

Snarf (Obtener): Se le llama *Snarf* cuando la caché obtiene los datos de la transacción. Esto permite a la caché estar actualizada con los datos de la memoria principal y de esta manera mantiene la consistencia [Intel(2010)].

Otros términos son comúnmente utilizados para describir las inconsistencias en los datos de la caché, describiéndolos:

Dirty Data (Datos Sucios) Cuando los datos son modificados dentro de la caché pero no modificados en la memoria principal.

Stale Data (Datos Viejos) Cuando los datos son modificados dentro de la memoria principal, pero la caché no contiene esas modificaciones.

El autor comenta que en el caso de la presente tesis, como se refiere a los datos históricos, estos no cambian dentro de la base de datos. Por esta razón, el mecanismo de caché es de solo lectura, además de almacenar solamente los datos que no se encuentren propensos a modificación por el servidor de históricos.

1.3. Caché en arquitectura cliente-servidor

Un elemento crucial para mantener altas prestaciones y escalabilidad en sistemas de base de datos cliente-servidor, es utilizar los recursos computacionales y de almacenamiento de los clientes. Por esta razón, mucho de estos sistemas están basados en la arquitectura *Data-Shipping*. En este tipo de arquitectura, el proceso de pedido es ejecutado muchas veces por el cliente y la copia de los datos son procesados por demanda desde los servidores. En el orden de mantener minimizada la latencia⁴ y la necesidad de una interacción futura con el servidor, mucho de los sistemas *Data-Shipping* usan la memoria principal del cliente y/o la Unidad de Disco Duro (HDD, por sus siglas en inglés), para el caché de los datos que son recibidos por parte del servidor para un posterior uso de estos [Tan(1996)].

Cuando la caché es incorporada dentro de la arquitectura de *Data-Shipping*, los servidores son usados para servir los *Miss*, de esta manera, la interacción cliente-servidor es catalogada como *fault-driven*⁵. La relación existente entre el cliente y el servidor, en este caso, es similar a la que existe entre, el administrador de *buffer* de una base de datos y el administrador de discos de un sistema de base de datos centralizado.

Las técnicas utilizadas para manejar el caché en los clientes, en los sistemas *Data-Shipping*, están relacionadas con las desarrolladas para los sistemas de administración de *buffer* de una base de datos, en los sistemas tradicionales. La caché de un cliente es administrada como un conjunto de elementos individuales, típicamente “páginas” o “tuplas”. Un elemento individual puede ser ubicado en la caché, realizando una búsqueda, utilizando sus identificadores o escaneando el contenido interno de la caché [Tan(1996)].

⁴Se refiere al tiempo que transcurre entre el pedido y la respuesta

⁵Terminología utilizada cuando se indica que se realiza una conexión al servidor solamente cuando ocurre un *Miss* en la caché.

Como en los controladores de *buffer*, una de las principales responsabilidades de un controlador de caché, en el cliente, es la de determinar que datos deben ser retenido dentro de la caché, siempre contando con un limitado espacio. Estas decisiones se realizan usando una política de reemplazo (Sección 1.5), para cada elemento es asignado un valor y cuando se necesita contar con más espacios, utilizando esa información y determinados algoritmos, son seleccionados los candidatos para reemplazar [Tan(1996)].

Analizando lo escrito por Kaladhar Voruganti, M. Tamer Özsu y Ronald C. Unrau [Voruganti(2004)], se puede entender que la arquitectura *Data-Shipping* reduce la latencia de la red e incrementa la utilización de los recursos del cliente. El Administrador de Base de Datos Orientadas a Objetos (ODBMS, por sus siglas en inglés), sistemas de ficheros, sistemas de datos móviles, servidores web, todos utilizan alguna variante de *Data-Shipping*. A pesar de las investigaciones realizadas durante décadas todavía existen controversias acerca de el tipo de implementación de *Data-Shipping* a usar.

A criterio del autor, independientemente de las discrepancias generadas, el desarrollo se basa en la utilización de la caché, alojado en la PC cliente, como un mecanismo transparente a las consultas generadas por la aplicación que hace uso de la libHDA.

1.4. Almacenamiento de la caché

Los datos de la caché pueden ser almacenados en diferentes dispositivos, en la propia RAM o en un HDD. La pirámide describiendo la relación entre velocidad y capacidad de almacenamiento se puede observar en la Figura 1.6.

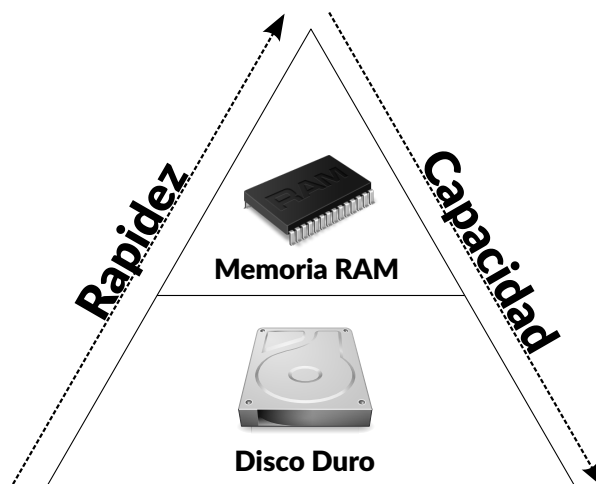


Figura 1.6. Relación entre la velocidad y la capacidad de almacenamiento entre RAM y HDD

1.4.1. Serialización de datos en el HDD

Serializar es el método que se utiliza para representar un dato en un arreglo de bytes, el cual generalmente se utiliza para transmitir de un medio a otro utilizando un canal determinado. Entre los usos más comunes está el de guardar datos que se encuentren en RAM hacia el HDD. Para dicha tarea se han implementado diversas bibliotecas para abstraer al programador del proceso en sí, brindando una Interfaz de Programación de Aplicaciones (API, por sus siglas en inglés) en los lenguajes de programación más usados, como C++, Java, C#, entre otros. A continuación se enuncian algunas bibliotecas utilizadas para esta tarea.

Boost::Serialization

Se puede obtener información más detallada desde su sitio web oficial [[Boost\(2004\)](#)], esta biblioteca forma parte de las bibliotecas de *Boost* [[Boost\(2004\)](#)]. Cumple con una serie de características dictadas en [[Boost\(2004\)](#)] con las que todo sistema de serialización debería contar (según *Boost*), para el lenguaje C++, estas son:

1. Portabilidad del código, dependiendo solamente de las facilidades del Instituto Nacional Estadounidense de Estándares (ANSI, por sus siglas en inglés) C++.
2. Economía de código, explotar las características de C++ como la Información de Tipos en Ejecución (RTTI, por sus siglas en inglés), plantillas, múltiples herencias, etc. donde sean apropiados para crear código fuente corto y simple de usar.
3. Versionado diferente por cada definición de clase. Esto es, cuando la definición de una clase cambia, los ficheros antiguos pueden importarse aun a la nueva versión de la clase.
4. Salva y restauración de punteros profundamente, Esto es, a la hora de salvar o restaurar un puntero, lo hace al dato al cual apunta.
5. Restauración correcta de punteros a datos compartidos.
6. Serialización de contenedores de la biblioteca estándar de C++, además de otras estructuras comúnmente utilizadas.
7. Portabilidad de los datos, los datos almacenados en una plataforma deben poder ser leídos en otra plataforma.
8. Especificación ortogonal de la serialización de las clases y formatos de archivos. Esto es, cualquier formato de fichero debe restaurar cualquier estructura de datos de C++ sin tener que alterar la serialización de ninguna clase.
9. No intrusivo. Permitir la serialización en clases sin alterar código de esta.

10. La interfaz del fichero debe ser lo suficientemente simple para permitir la creación de nuevos tipos de ficheros.
11. La interfaz del fichero debe ser lo suficientemente entendible para permitir la creación de un archivo que presente los datos serializados en XML.

Protobuff

Según Google[®] [[Google\(2014a\)](#)], empresa tras el desarrollo de esta biblioteca, protobuff es la implementación de “*Protocol Buffers*”, el cual es un flexible, eficiente y automatizado mecanismo para la serialización de datos estructurados –como el Lenguaje Extensible de Marcas (XML, por sus siglas en inglés), pero más pequeño, rápido y simple –. Se define la estructura de los datos a serializar utilizando un formato específico de *Protobuff*, luego se utiliza un código fuente generado a través de sus herramientas, que le permite escribir y leer este tipo de datos por una variedad de canales, usando una variedad de lenguajes de programación. Incluso se puede actualizar la estructura de los datos sin corromper los programas desplegados que utilizan el viejo formato. Presenta varias ventajas con respecto al XML, como son:

- Es más simple.
- Es de 3 a 10 veces más pequeño.
- Es de 20 a 100 veces más rápido.
- Es menos ambiguo.
- Genera clases de acceso a datos que son fáciles de usar por el programador.

Algunas desventajas, vistas en Google[®] [[Google\(2014b\)](#)], se muestran a continuación:

Flujo de múltiples mensajes: El formato de *Protobuff* no se auto delimita, por lo que no es posible determinar cuando un mensaje termina.

Grandes conjunto de datos: *Protobuff* no está diseñado para manejar grandes mensajes. Si se trabaja con mensajes superiores a 1 megabyte cada uno, es tiempo de considerar el uso de una estrategia alternativa.

Tipos Unión: A veces es necesario enviar datos que pueden ser de uno u otro tipo, pero el analizador de *Protobuff* necesariamente no puede determinar el tipo de un dato basándose en su contenido.

Mensajes auto-descriptivos: *Protobuff* no salva descripciones para sus propios tipos, por lo que si solamente se tiene un mensaje en bruto sin el correspondiente fichero que especifica la estructura, es muy difícil extraer datos útiles.

MessagePack

Como comenta Sadayuki Furuhashi [[Furuhashi\(2013\)](#)], en el sitio oficial de esta solución, MessagePack es un formato de serialización binaria eficiente. Permite que intercambies datos a través de múltiples lenguajes de programación, al igual que JSON, pero más rápido y más pequeño. Los enteros pequeños son codificados dentro de un único byte, además de que las típicas cadenas cortas requieren solamente un byte adicional además de ella propia.

Algunas limitaciones, por Sadayuki Furuhashi [[itiut\(2014\)](#)]:

- El valor de los enteros está limitados a encontrarse en el rango -2^{63} y $2^{64} - 1$ ⁶
- El valor del tipo “Float” está sujeto a la norma IEEE 754 simple o doble precisión.
- El tamaño máximo de un solo objeto “Binario” es de $2^{32} - 1$ (4Gbytes).
- El tamaño máximo de un solo objeto de tipo “Cadena” es $2^{32} - 1$ (4Gbytes).
- Los objetos de tipo “Cadena” pueden contener secuencias de caracteres inválidas y el comportamiento del deserializador depende de la implementación realizada por el programador.
- El número máximo de elementos en un tipo “Arreglo” es $2^{32} - 1$ (4'294'967'295).
- El número máximo de asociaciones llave-valor de un tipo “Mapa” es $2^{32} - 1$ (4'294'967'295).

s11n

Teniendo en cuenta a Stephan Beal [[Beal\(2013b\)](#)] s11n⁷ fue creada en 2004 [[Beal\(2013a\)](#)] y es una biblioteca para la fácil serialización de una amplia variedad de objetos, incluyendo los Tipos de Datos Planos (POD, por sus siglas en inglés). Es un proyecto *Open Source* orientado a C⁸ y C++. Según Stephan Beal [[Beal\(2012a\)](#)] soporta literalmente ciento de millones de combinaciones de los contenedores de la biblioteca estándar de C++ sin codificación adicional, además de que incluye soporta para varios formatos [[Beal\(2012c\)](#)]:

funtxt: Gramática simple basada en texto plano.

funxml: Dialecto básico de XML con algunas extensiones de su compilación estándar.

simplexml: Basado en funxml, pero optimizado para datos pequeños, como números.

⁶Para tener una representación de lo que significa este gran número, si se toma este número como cantidad de segundos, esto equivale a 593'066'617 Milenios

⁷El nombre proviene de “serialization”, tomando la primera y última letra y ubicando entre ellas la cantidad de caracteres que hay (11)

⁸En versiones recientes se incluyó el soporte de C, pues solamente estaba para C++

compact: Formato binario.

wesnoth: Otro serializador XML que usa los nodos de manera `[nodename] ... [/nodename]`.

Principales características [Beal(2012b)]:

- Provee de interfaces fáciles de usar, fácil de integrar y de bajo mantenimiento para la serialización de una variedad de tipos de C++.
- El soporte de serialización puede ser adicionado sin necesidad de modificar los tipos⁹.
- Sin necesidades adicionales cuenta con soporte para todos los contenedores de la biblioteca estándar de C++.
- Puede extenderse fácilmente para el soporte de tipos personalizados.
- Cuenta con soporte nativo de varios formatos de serialización, además de brindar la posibilidad de extender la biblioteca con otros.
- La biblioteca utiliza solamente la norma de la Organización Internacional para la Normalización (ISO, por sus siglas en inglés) de C++, lo que la permite ser compilada por la inmensa mayoría de los compiladores que soportan esta norma.
- La biblioteca trae consigo una gran cantidad de documentación.

Otras bibliotecas

Bibliotecas no referenciadas en el presente trabajo:

Microsoft® Foundation Class (MFC, por sus siglas en inglés): Contienen un mecanismo de serialización para permitir que los objetos persistan entre las ejecuciones de su programa [Microsoft(2013)].

CommonC++: Según GNU No es Unix (GNU, por sus siglas en inglés) [GNU(2009)], es una biblioteca de clases de C++ que abstrae de una manera portable varios servicios del sistema. Entre los módulos que trae consigo, contiene el de serialización de objetos.

Eternity: Analizando lo expuesto en [jwl(2010)] y [Santi(2013)], es un pequeño *framework* compatible con la biblioteca estándar de C++ para almacenar y obtener cualquier tipo de clase (adicionalmente clases plantilla). Puede realizarlo utilizando serialización binaria o XML.

⁹Se toma como significado de tipo, cualquier clase, estructura, etc. que puede ser denominada en C/C++

1.5. Políticas de reemplazo en la caché

El almacenamiento de los datos en la caché es finito, por lo que en algún momento se llega al límite de almacenamiento de los datos temporales, es aquí donde entran en función las políticas de reemplazo en la caché. Estos mecanismos no son más que los algoritmos, métodos, procedimientos utilizados para seleccionar el mejor candidato a reemplazar, se explican algunos de estos en los epígrafes siguientes.

De la elección de estos algoritmos para la implementación, en especial depende el rendimiento general del sistema que utiliza el mecanismo de caché. Cada algoritmo contiene características adaptadas a cada escenario en específico, por lo que se reafirma la importancia de la situación problemática presente. Aunque todos los algoritmos hacen su trabajo para cualquiera de los problemas, lo necesario es la eficiencia y acortar el costo del pedido del cliente. El tiempo medio de acceso en memoria es [Drepper(2007)]:

$$T = m * T_m + (1 - m) * T_h + E$$

donde:

T = Tiempo medio de acceso al elemento.

m = Probabilidad de fallo (**Miss**) = 1 - (probabilidad de acierto (**Hit**)).

T_m = Tiempo para hacer un acceso a memoria¹⁰.

T_h = Latencia, tiempo para acceder al elemento en caché cuando ha habido un acierto.

E = Efectos secundarios, como colas mantenidas por los multiprocesadores.

1.5.1. Algoritmo Bélády's

El algoritmo de reemplazo de caché más eficiente debe ser aquel que selecciona como candidato para eliminar, al que no será necesario en el futuro. Este algoritmo óptimo es conocido como el de "László Bélády's" o "Clarividencia". Debido a que es generalmente improbable saber en un futuro que datos se utilizarán, ni cuando se utilizarán, es prácticamente imposible la implementación de dicho procedimiento. Se utiliza principalmente para medir el rendimiento de los demás algoritmos. Se puede ver en un ejemplo utilizando un límite de la caché de 3 elementos en la Figura 1.7

¹⁰En el caso pertinente del presente trabajo sería al servidor de base de datos

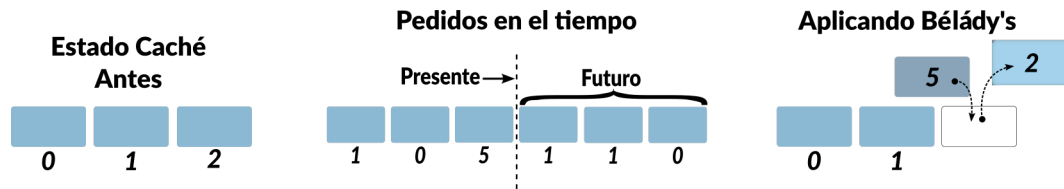


Figura 1.7. Algoritmo Bélády's

1.5.2. Algoritmo Menos Recientemente Usado (LRU, por sus siglas en inglés)

Este algoritmo mantiene una variable con el tiempo en que fue accedido el dato en el caché, de esta manera, en el proceso de selección del candidato a ser reemplazado, se ubica al de menor valor de dicha variable. Resulta en que solo los datos más modernos (actualizados) se mantengan en la caché. Existe una implementación alternativa, la cual optimiza al LRU llamada *LRU/k*, la cual utiliza un método de asignarle prioridad al elemento más reciente [Shasha(1994)].

Se aprecia un ejemplo en la Figura 1.8 en la cual se toma en cuenta el tiempo en que fue accedido desde que se creó la caché.

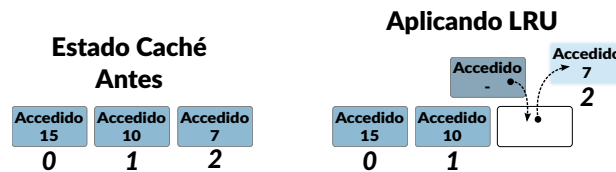


Figura 1.8. Algoritmo LRU

1.5.3. Algoritmo Más Recientemente Usado (MRU)

Algoritmo que opera exactamente como el opuesto del LRU. Utiliza como criterio de selección al que se haya accedido más recientemente. Este algoritmo es más utilizado cuando los datos más viejos son los necesarios. Se observa un ejemplo en la Figura 1.9 en la cual se toma en cuenta el tiempo en que fue accedido desde que se creó la caché.



Figura 1.9. Algoritmo MRU

1.5.4. Algoritmo Menos Frecuentemente Usado (LFU, por sus siglas en inglés)

Algoritmo que utiliza (al igual que LRU y MRU) datos adicionales dentro de la caché, en este caso un contador de referencias (cantidad de veces que se ha accedido) que se mantiene junto a los datos en sí. El que menos referencias reciba es el seleccionado para ser reemplazado por el dato entrante. La manera más simple de implementar este tipo de algoritmo es agregar un dato adicional por cada elemento, el cual es incrementado por cada referencia realizada [Matani(2010)] [Ding(2011)]. Un ejemplo en la Figura 1.10.

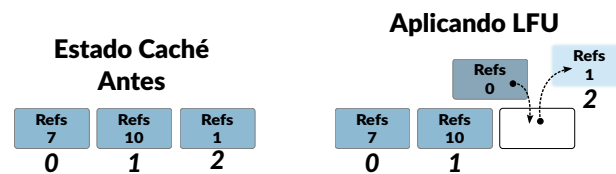


Figura 1.10. Algoritmo LFU

Presenta algunos aspectos negativos, dentro de ellos se encuentran los siguientes [Ding(2011)]:

- Si se cuenta con un elemento que es accedido muchas veces en un corto período de tiempo, el cual no se accede luego por una larga etapa, es poco probable que se seleccione para ser reciclado cuando se ejecute el algoritmo. Esto incide en la eliminación de elementos más utilizados que este.
- También se evidencia que los datos que son recientemente insertados comienzan con un contador de referencias bajo, por lo que son muy propensos a ser seleccionados para reciclado sin tener en cuenta futuros usos.

1.5.5. Otros algoritmos

Son muchos los algoritmos implementados en la actualidad para darle solución al problema del remplazo de los elementos dentro de la caché. Entre estos se pueden mencionar:

- Algoritmo Remplazo Aleatorio (RR, por sus siglas en inglés). Utilizado principalmente en los microprocesadores ARM[®] Cortex [ARM(2000)].
- Algoritmos basados en contadores: AIP y LvP [Solihin(2005)].
- Algoritmo de predicción con referencia de intervalo [Emer(2006)].
- Algoritmo Remplazo de Caché Auto Configurable y de Baja Sobrecarga (ARC, por sus siglas en inglés) [Modha(2003)].
- Algoritmo de remplazo utilizando múltiples colas para la caché de segundo nivel [Philbin(2001)].

- Algoritmo Pseudo-LRU (PLRU).
- Algoritmo Segmented-LRU (SLRU).

1.6. Metodología de desarrollo

Las metodologías para el desarrollo de *software* definen reglas a cumplir para lograr un proceso disciplinado sobre el desarrollo de *software*, con el fin de hacerlo más predecible y eficiente. La calidad del *software* se ve afectada directamente de manera positiva en cada una de sus fases de desarrollo, haciendo énfasis en la calidad y menor tiempo de construcción del *software*; o citando a Ph. D. Roger S. Pressman [[Pressman\(2010\)](#)] "(...) producir lo esperado en el tiempo esperado y con el coste esperado (...)".

En la UCI, se utilizó la metodología Proceso Unificado Ágil (AUP, por sus siglas en inglés) para definir una variante (AUP-UCI) ha ser utilizada por sus proyectos productivos. De esta manera, se adapta a los procesos personalizados de la Universidad. El presente trabajo se integra al proceso de desarrollo del SCADA Galba, el cual utiliza esta nueva metodología, por lo cual se hace uso de ella para su desarrollo.

1.6.1. AUP

AUP es una versión simplificada de Proceso Unificado de Rational (RUP, por sus siglas en inglés) desarrollada por Scott Ambler [[Waters\(2008\)](#)]. Este describe una manera simple y fácil de entender la forma de desarrollar aplicaciones de *software* de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. Aplica técnicas incluyendo Desarrollo Guiado por Pruebas (TDD, por sus siglas en inglés), Modelado Ágil (AM, por sus siglas en inglés), administración de cambios ágiles y refactorización de base de datos para aumentar la productividad.

Proceso Unificado (UP, por sus siglas en inglés) es un marco de desarrollo de *software* iterativo e incremental. A menudo es considerado como un proceso altamente ceremonioso porque especifica muchas actividades y artefactos involucrados en el desarrollo de un proyecto de *software*. Dado que es un marco de procesos, puede ser adaptado y la más conocida es RUP de International Business Machines Corporation (IBM, por sus siglas en inglés) [[Flores\(2009\)](#)].

El proceso AUP establece un Modelo más simple que el que aparece en RUP por lo que reúne en una única disciplina [[Edeki\(2013\)](#)]:

- El Modelado de Negocio.
- El Modelado de Requisitos y Análisis.

- El Diseño

De las cuatro fases que propone AUP (Inicio, Elaboración, Construcción, Transición) se decide para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando el objetivo de la misma, se unifican las restantes tres fases de AUP en una sola, a la que se llama Ejecución y se agrega una fase de Cierre.

AUP propone siete disciplinas (Modelo, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyecto y Entorno), se decide para el ciclo de vida de los proyectos de la UCI tener ocho disciplinas, pero a un nivel más atómico que el definido en AUP. Los flujos de trabajos: Modelado de negocio, Requisitos y Análisis y diseño en AUP están unidos en la disciplina Modelo, en la variación para la UCI se consideran a cada uno de ellos disciplinas.

Principios de AUP AUP está basada en los siguientes principios [[Ambler\(2006\)](#)]:

- *El personal sabe lo que están haciendo.* La gente no va a leer la documentación de los procesos en detalle, sino que quieren una orientación de alto nivel y/o formación de vez en cuando. El producto AUP proporciona enlaces a muchos de los detalles si uno está interesado, pero no obliga seguir los detalles.
- *Simplicidad.* Todo se describe concisamente usando unas páginas, no miles de páginas.
- *Centrarse en las actividades importantes.* La atención se centra en las actividades que realmente cuentan.
- *Independencia de herramienta.* Poder usar cualquier herramienta que desee utilizar con la AUP. Es recomendable utilizar herramientas que mejor se adapten para el trabajo, que a menudo son herramientas simples o incluso herramientas de código abierto.
- *Querer adaptar este producto para satisfacer sus propias necesidades.* El producto AUP es fácil de manejar a través de cualquier herramienta de edición de Lenguaje de Marcado de Hipertexto (HTML, por sus siglas en inglés). No necesita comprar una herramienta especial, o tomar un curso, para adaptar el AUP.

Ventajas de AUP

El personal sabe lo que está haciendo: No obliga a conocer detalles.

Simplicidad: Apuntes concisos.

Agilidad: Procesos simplificados del RUP.

Centrarse en actividades de alto valor: Esenciales para el desarrollo.

Herramientas independientes: A disposición del usuario.

Fácil adaptación de este producto: De fácil acomodo (HTML)

Desventajas de AUP

- Es un producto muy pesado en relación a RUP.
- Como es un proceso simplificado, muchos desarrolladores eligen trabajar con RUP, por tener a su disposición más detalles acerca de un proceso [[Flores\(2009\)](#)].

1.7. Herramientas y tecnologías

A continuación se brinda una panorámica acerca de las herramientas y tecnologías utilizadas para el desarrollo de la biblioteca (libHDA) del SCADA Galba. Estas continúan siendo utilizadas en su inmensa mayoría, puesto que el desarrollo del mecanismo es dentro de dicha biblioteca, para evitar aumentar el tiempo requerido para el desarrollo, así como la curva de aprendizaje. Se realizará la comparación acerca del cambio de tecnologías, brindando (a través de la experiencia del desarrollador y bibliografía) la opinión acerca de dicho cambio.

1.7.1. Lenguaje de modelado

Lenguaje de Modelado Unificado (UML, por sus siglas en inglés) 2.0

Tomando en cuenta a Russ Miles y Kim Hamilton [[Hamilton\(2006\)](#)], UML es un lenguaje estándar en el análisis y diseño de sistemas de cómputo que permite especificar, visualizar, construir y documentar todos los elementos que forman un sistema de *software*, desde una perspectiva orientada a objetos. Este modelado visual es independiente del lenguaje de implementación, los diseños realizados utilizando UML se pueden implementar principalmente en lenguajes orientados a objetos. Ampliamente utilizado para dar una idea general al programador en un modelo no tan abstracto ni tan específico de implementación, se podría comentar que sería una analogía para sistemas orientados a objetos como lo es el pseudocódigo de los algoritmos.

1.7.2. Sistemas homólogos

La presente tesis, define un mecanismo para disminuir el tiempo de acceso a los datos consultados por la biblioteca de acceso a datos del SCADA Galba, su implementación se encuentra dentro de esta biblioteca; en la búsqueda de sistemas que realicen la misma tarea se debe precisar la incursión dentro de otras bibliotecas de acceso a datos en los componentes de históricos reconocidos, dígame WinCC[®], Wonderware[®], entre otros; los cuales son productos comerciales

líderes en el mercado, que no brindan código fuente. De esta manera, no se pueden analizar estos sistemas que contendrían bibliotecas de acceso a datos homólogas que se desconoce si implementan dicho mecanismo de optimización.

De esta manera, se opta por realizar un sistema de caché, específicamente del tipo “Data-Shipping” que utilizaría más los recursos del cliente para aliviar el servidor, disminuyendo la latencia de obtención de los datos entre la aplicación cliente y el servidor (base de datos de históricos).

1.7.3. Herramienta para el modelado de diagramas

Visual Paradigm para UML CE 11.0

En el sitio oficial de Visual Paradigm [[Paradigm\(2011\)](#)] se muestra una definición que la define como una herramienta Ingeniería de Software Asistida por Computadora (CASE, por sus siglas en inglés) que soporta UML, Lenguaje de Modelado de Sistemas (SysML, por sus siglas en inglés) y Modelo y Notación de Procesos de Negocio (BPMN, por sus siglas en inglés). Además del soporte de modelado, provee generación de reportes y capacidades de ingeniería de código fuente, en las cuales se incluye generación de diagramas UML a partir de un *software* codificado, así como la exportación del modelo UML hacia varios lenguajes de implementación, tales como C++, Java, C#, entre otros. Esta solución presenta varias licencias para su uso, las cuales se pueden observar en el apartado de compra del sitio oficial [[Paradigm\(2011\)](#)], entre las que se encuentran:

- Modelador, con un costo de US \$ 99
- Estándar, con un costo de US \$ 299
- Profesional, con un costo de US \$ 699
- Empresarial, con un costo de US \$ 1 399

Existe un caso de versión especial llamada “Community Edition”(Edición comunitaria) que permite la utilización sin fines comerciales.

1.7.4. Lenguaje de programación

C++

C++ es un lenguaje compilado [[Stroustrup\(2013\)](#)] de programación de propósito general creado por Bjarne Stroustrup en el año 1983. Tiene características imperativas, orientada a objetos y programación genérica; brindando además soporte para el manejo a bajo nivel de la memoria del sistema. Ha sido diseñado con un sesgo hacia la programación de sistemas (por ejemplo para

el uso en dispositivos empotrados¹¹ o núcleos de sistemas operativos), con la eficiencia, rendimiento y flexibilidad de uso como sus requerimientos de diseño. C++ se puede encontrar también muy usado en otros contextos, incluyendo aplicaciones de escritorio, servidores (web, de base de datos, etc.), aplicaciones donde el rendimiento es crítico (administradores telefónicos, pruebas espaciales, sistemas SCADA, etc.), además de *software* para entretenimiento [Stroustrup(2014)]. Está estandarizado por la ISO [ISO(2014)].

Lenguaje de Consulta Estructurado (SQL, por sus siglas en inglés)

SQL es un lenguaje de propósito especial diseñado para administrar los datos en un Sistema de Administración de Base de Datos Relacional (RDBMS, por sus siglas en inglés) o para procesamiento de flujos en los Sistema de Administración de Flujos de Datos Relacional (RDSMS, por sus siglas en inglés). Es uno de los primeros lenguajes comerciales para el modelo relacionar de Edgar F. Codd, como describió en su artículo en 1970 “A Relational Model of Data for Large Shared Data Banks ”[Codd(1970)]. Se vuelve una norma del ANSI en 1986 y del ISO en 1987 [IBM(2011)]. Desde entonces la norma ha sido revisada para incluir un gran conjunto de funcionalidades. Aunque existan dichas normas, la mayoría del código SQL no es completamente portable¹² a lo largo de los diferentes sistemas de base de datos sin ajustes.

1.7.5. Entorno Integrado de Desarrollo (IDE, por sus siglas en inglés)

Eclipse Kepler 3.9

Eclipse [Eclipse(2014)] es un IDE desarrollado en su mayoría en Java por la *Eclipse Foundation*. Contiene una base de espacios de trabajo y un extensible sistema de complementos para personalizar el entorno. Aunque fundamentalmente está orientado a desarrollar aplicaciones que utilizan tecnologías de Java™, a través de las extensiones se puede prácticamente realizar cualquier tipo de aplicación si se cuenta con el complemento para ello. Para C++ se cuenta con el complemento CDT. Este entorno es el utilizado por la línea de históricos.

KDevelop 4.7

KDevelop es un IDE gratis y de código abierto para GNU/Linux, Solaris®, FreeBSD®, Apple® MacOSX® y otros sistemas operativos. Su principal objetivo es ayudar a los programadores del lenguaje C/C++(Sección 1.7.4), pero a través de su arquitectura basada en complementos se puede agregar soporte para disímiles lenguajes, como Java, PHP, Python, entre otros. Está basado en

¹¹Sistema computacional con una función dedicada sin tener un gran sistema mecánico o electrónico

¹²Se refiere a que pueden existir extensiones al lenguaje en un sistema que no estén en otro

las bibliotecas de KDevPlatform¹³, KDE¹⁴ y Qt¹⁵; desarrollándose desde 1998. Una de las funcionalidades por la que más se destaca es por el motor de análisis de código de C/C++ que permite al desarrollador una gran fluidez a la hora del trabajo con aplicaciones que se desarrollen en este lenguaje, permitiéndole navegar entre clases, generar y actualizar las definiciones de las clases, métodos y variables, control de versiones con Git¹⁶, SVN¹⁷, etc. [[KDevelop\(2014\)](#)].

1.7.6. Bibliotecas de desarrollo

Boost Libraries

Boost provee bibliotecas de C++ portables, libres y muy revisadas. Está desarrollada con un enfoque de que contenga toda la mayor cantidad de utilidades para los programadores y que sea compatible completamente con la STL¹⁸. Uno de los puntos que demuestran el grado de estandarización de este conjunto de bibliotecas es que están incluidas en el comité estándar de C++ [[ISO\(2014\)](#)] por lo que sus bibliotecas están en revisión para cada salida de un nuevo estándar de este lenguaje. Brinda soporte para casi cualquier sistema operativo moderno, incluyendo las variantes de UNIX[®] y Microsoft[®] Windows[™] [[Boost\(2014\)](#)].

1.7.7. Sistema de Gestión de base de datos

PostgreSQL 9.1

Gestor de base de datos orientada a objetos, muy conocido y usado en entornos de *software* libre. Puede funcionar en múltiples plataformas. Cuenta con un rico conjunto de tipos de datos, permitiendo además su extensión mediante tipos y operadores definidos y programados por el usuario. Su administración se basa en usuarios (llamados roles) y privilegios. Es altamente confiable en cuanto a estabilidad se refiere. Puede extenderse con librerías externas para soportar características personalizadas [[Postgres\(2014\)](#)].

¹³Conjunto de bibliotecas para el apoyo a la realización de IDEs.

¹⁴Entorno de escritorio principalmente para GNU/Linux.

¹⁵Framework de desarrollo que contiene una gran cantidad de módulos de apoyo al desarrollador, como Red, Visual, Gráficos 3D, entre otros.

¹⁶Sistema de control de versiones creado por Linus Torvalds en el 2005 principalmente para el núcleo de GNU/Linux, muy rápido y escalable.

¹⁷Sistemas de control de versiones creado por Apache Foundation en el 2000

¹⁸Es el conjunto de cabeceras, cuya implementación depende del sistema operativo, que por defecto trae C++ para su uso.

1.8. Consideraciones del capítulo

Se demuestra la necesidad de implementar un mecanismo de caché como solución al problema planteado, que almacenará temporalmente los datos consultados por la libHDA para luego ser reutilizados en futuros pedidos. De esta forma, se evitan consultas redundantes a la base de datos y por ende, sobrecarga en el tiempo de respuesta. Entre los dos IDE expuestos, se utiliza KDevelop, tomando en cuenta varios aspectos basados en la experiencia propia del autor en el uso de estas herramientas.

CAPÍTULO 2

PROPUESTA DE SOLUCIÓN

2.1. Introducción

En este capítulo se definen los requisitos tanto funcionales como no funcionales de la solución en cuestión. Se define el modelo de dominio, se describen los estilos arquitectónicos, patrones de diseño aplicados, diagramas de clases del diseño y el modelo de despliegue. Se toma en referencia los conceptos y términos descritos en el capítulo anterior, regido por la metodología AUP-UCI.

2.2. Descripción del proceso a automatizar

La libHDA permite a las aplicaciones que la utilicen solicitar datos al servidor de base de datos de históricos, ya sean puntos, alarmas, etc. La biblioteca es la encargada de acceder directamente a este servidor y retornarle al que lo requiera dichos datos de manera ordenada.

Durante el proceso de obtener los datos, la biblioteca no realiza un guardado temporal de estos en ningún lugar de almacenamiento, por lo que no puede reutilizarlos y por tanto debe realizar el pedido completo al servidor de base de datos por cada consulta que se haga.

Una vez desarrollado el mecanismo de caché para la libHDA, permitirá que se le configure la cantidad en megabytes de capacidad a tomar de RAM y HDD, de esta manera se almacenan temporalmente los datos obtenidos y así poder responder a pedidos futuros y con esto disminuir el tiempo de respuesta ante pedidos con intervalos de tiempo solapados.

2.3. Modelo de dominio

La etapa orientada a objetos, esencial en el análisis o investigación, es la descomposición de un dominio de interés en clases conceptuales individuales u objetos. Un **modelo de dominio** es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés. También se les denomina **modelos conceptuales**, **modelo de objetos del dominio** y **modelos de objetos de análisis** [Larman(2003)].

2.3.1. Diagrama de Clases del Dominio

Con el objetivo de describir y expresar el contexto en que se desarrolla el módulo, en la Figura 2.1 se exponen los conceptos que son pertinentes para la presente investigación.

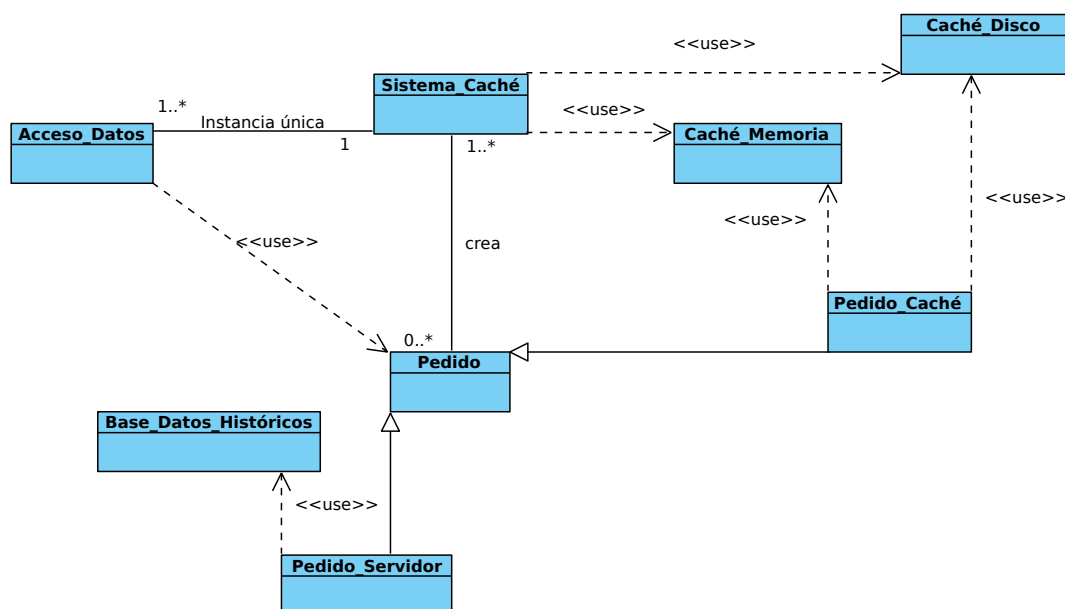


Figura 2.1. Diagrama de clases del dominio

2.3.2. Descripción de las clases del modelo de dominio

Para una mayor claridad y comprensión del Modelo de Dominio del negocio, se proporciona una breve descripción de las clases que lo integran:

Base_Datos_Históricos: Indica la base de datos donde se encuentran alojados todos los datos que pueden ser consultados por la biblioteca de acceso a datos.

Acceso_Datos: Asocia a la interfaz que muestra la biblioteca para su uso por terceras aplicaciones, permitiendo realizar las consultas pertinentes a la base de datos de histórico.

Sistema_Caché: Concepto asociado al encargado de gestionar todo el sistema de almacenamiento y consulta de la caché, además de la generación de los pedidos para ser usado por **Acceso_Datos**.

Pedido: Se Refiere a un pedido generado por **Acceso_Datos** y tratado por **Sistema_Caché** para darle el camino¹ por donde obtendrá los datos.

Pedido_Servidor: Indica un pedido que hará uso de **Base_Datos_Históricos** para obtener y entregar los datos.

Caché_Memoria: Se refiere al lugar en la RAM donde se encuentran alojados todos los datos temporales.

Caché_Disco: Se refiere al lugar en el HDD donde se encuentran alojados todos los datos temporales.

Pedido_Caché: Indica un pedido que hará uso de **Caché_Disco** o **Caché_Memoria** para obtener y entregar los datos.

2.4. Especificación de requisitos de *software*

Los requisitos para un sistema son la descripción de los servicios proporcionados por el sistema y sus restricciones operativas. Estos requisitos reflejan la necesidad de los clientes de un sistema que ayude a resolver algún determinado problema como el control de un dispositivo, hacer un pedido o encontrar información. El proceso de descubrir, analizar, documentar y verificar esos servicios y restricciones es denominado ingeniería de requisitos [[Sommerville\(2005\)](#)].

2.4.1. Requisitos funcionales

Los requisitos funcionales de un sistema son los que describen las capacidades o funciones que el sistema debe cumplir, los servicios que de él se esperan, o los que proveerá [[Sommerville\(2005\)](#)].

Historias de Usuario

Una historia de usuario es una representación de un requisito de *software* escrito en una o dos frases utilizando el lenguaje común del usuario. Son utilizadas en las metodologías de desarrollo ágiles para la especificación de requisitos. Son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Permiten responder rápidamente a los requisitos cambiantes [[Suaza\(2013\)](#)].

¹Se refiere a determinar el origen de los datos, ya sea de la caché o del servidor.

2.4.2. Descripción de las Historias de Usuario

Tabla 2.1. RF1 Almacenamiento en RAM de los datos temporales.

Historia de Usuario	
Número H1	Nombre del requisito Almacenamiento en RAM de los datos temporales.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Alta	Tiempo estimado 1271 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 571 horas
Descripción Todos los datos pedidos por la aplicación que hace uso de la libHDA deben ser almacenados en la RAM mientras quede espacio disponible para dicha tarea.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.2. RF1.1 Almacenamiento en RAM de puntos analógicos.

Historia de Usuario	
Número H1.1	Nombre del requisito Almacenamiento en RAM de puntos analógicos.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Alta	Tiempo estimado 400 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 140 horas
Descripción Todos los puntos analógicos pedidos por la aplicación que hace uso de la libHDA deben ser almacenados en la RAM mientras quede espacio disponible para dicha tarea.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.3. RF1.2 Almacenamiento en RAM de puntos digitales.

Historia de Usuario	
Número H1.2	Nombre del requisito Almacenamiento en RAM de puntos digitales.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Continúa en la próxima página	

Tabla 2.3 – Continuada desde la página anterior

Historia de Usuario	
Prioridad Alta	Tiempo estimado 400 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 160 horas
Descripción Todos los puntos digitales pedidos por la aplicación que hace uso de la libHDA deben ser almacenados en la RAM mientras quede espacio disponible para dicha tarea.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.4. RF1.3 Almacenamiento en RAM de logs del sistema.

Historia de Usuario	
Número H1.3	Nombre del requisito Almacenamiento en RAM de logs del sistema.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Alta	Tiempo estimado 471 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 271 horas
Descripción Todos los logs del sistema pedidos por la aplicación que hace uso de la libHDA deben ser almacenados en la RAM mientras quede espacio disponible para dicha tarea.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.5. RF2 Almacenamiento en HDD de los datos temporales.

Historia de Usuario	
Número H2	Nombre del requisito Almacenamiento en HDD de los datos temporales.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Alta	Tiempo estimado 1271 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 470 horas
Continúa en la próxima página	

Tabla 2.5 – Continuada desde la página anterior

Historia de Usuario	
Descripción	Al llegar al límite el almacenamiento en RAM, todos los datos seleccionados a través de alguno de los algoritmos de reemplazo, son almacenados en el HDD mientras quede espacio disponible.
Observaciones	N/A
Prototipo de interfaz	N/A

Tabla 2.6. RF2.1 Almacenamiento en HDD de puntos analógicos.

Historia de Usuario	
Número H2.1	Nombre del requisito Almacenamiento en HDD de puntos analógicos.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Alta	Tiempo estimado 400 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 135 horas
Descripción Al llegar al límite el almacenamiento en RAM, todos los puntos analógicos seleccionados a través de alguno de los algoritmos de reemplazo, son almacenados en el HDD mientras quede espacio disponible.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.7. RF2.2 Almacenamiento en HDD de puntos digitales.

Historia de Usuario	
Número H2.2	Nombre del requisito Almacenamiento en HDD de puntos digitales.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Alta	Tiempo estimado 400 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 135 horas
Descripción Al llegar al límite el almacenamiento en RAM, todos los puntos digitales seleccionados a través de alguno de los algoritmos de reemplazo, son almacenados en el HDD mientras quede espacio disponible.	
Continúa en la próxima página	

Tabla 2.7 – Continuada desde la página anterior

Historia de Usuario	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.8. RF2.3 Almacenamiento en HDD de logs del sistema.

Historia de Usuario	
Número H2.3	Nombre del requisito Almacenamiento en HDD de logs del sistema.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Alta	Tiempo estimado 471 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 200
Descripción Al llegar al límite el almacenamiento en RAM, todos los logs del sistema seleccionados a través de alguno de los algoritmos de reemplazo, son almacenados en el HDD mientras quede espacio disponible.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.9. RF3 Configurar espacio de almacenamiento en RAM.

Historia de Usuario	
Número H3	Nombre del requisito Configurar espacio de almacenamiento en RAM.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Alta	Tiempo estimado 100 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 20 horas
Descripción Establece el límite de almacenamiento en RAM.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.10. RF4 Configurar espacio de almacenamiento en HDD.

Historia de Usuario	
Número H4	Nombre del requisito Configurar espacio de almacenamiento en HDD.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Alta	Tiempo estimado 100 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 20 horas
Descripción Establece el límite de almacenamiento en HDD.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.11. RF5 Configurar la dirección lógica en el HDD donde serán almacenados los datos de la caché.

Historia de Usuario	
Número H5	Nombre del requisito Configurar la dirección lógica en el HDD donde serán almacenados los datos de la caché.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Alta	Tiempo estimado 5 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 2 horas
Descripción Se especifica el lugar en el sistema de ficheros donde se almacenarán los datos de la caché del HDD.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.12. RF6 Seleccionar el algoritmo de reemplazo para datos de la caché.

Historia de Usuario	
Número H6	Nombre del requisito Seleccionar el algoritmo de reemplazo para datos de la caché.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Alta	Tiempo estimado 288 horas
Continúa en la próxima página	

Tabla 2.12 – Continuada desde la página anterior

Historia de Usuario	
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 70 horas
Descripción Al llegar al límite de almacenamiento, tanto en RAM como en HDD, se debe utilizar un algoritmo (configurado) para reemplazar los datos seleccionados por los entrantes.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.13. RF7 Ordenar los datos devueltos por *timestamp*.

Historia de Usuario	
Número H7	Nombre del requisito Ordenar los datos devueltos por <i>timestamp</i> .
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Alta	Tiempo estimado 288 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 38 horas
Descripción Todos los datos pedidos por la aplicación que hace uso de la libHDA deben ser devueltos de manera ordenada por la propiedad <i>timestamp</i> del dato.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.14. RF8 Limpiar la caché.

Historia de Usuario	
Número H8	Nombre del requisito Limpiar la caché.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Baja	Tiempo estimado 20 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 10 horas
Descripción Elimina todos los datos de la caché.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.15. RF9 Habilitar y deshabilitar el mecanismo de caché.

Historia de Usuario	
Número H9	Nombre del requisito Habilitar y deshabilitar el mecanismo de caché.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Baja	Tiempo estimado 10 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 3 horas
Descripción Todos los datos pedidos por la aplicación que hace uso de la libHDA pasan directamente al cliente o se almacenan primeramente en la caché, según si se deshabilita o habilita respectivamente la caché.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.16. RF10 Consultar varios identificadores de un mismo tipo.

Historia de Usuario	
Número H10	Nombre del requisito Consultar varios identificadores de un mismo tipo.
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Media	Tiempo estimado 100 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 20 horas
Descripción Se adiciona la posibilidad de realizarle al mismo tiempo varios pedidos a la caché, evitando datos corruptos.	
Observaciones N/A	
Prototipo de interfaz N/A	

Tabla 2.17. RF11 Recuperarse ante fallas de conexión a la base de datos.

Historia de Usuario	
Número H11	Nombre del requisito Recuperarse ante fallas de conexión a las base de datos.
Continúa en la próxima página	

Tabla 2.17 – Continuada desde la página anterior

Historia de Usuario	
Programador Luis Felipe Domínguez Vega	Iteración asignada 1
Prioridad Media	Tiempo estimado 288 horas
Riesgo en desarrollo Problemas eléctricos o rotura de la estación de trabajo	Tiempo Real 30 horas
Descripción En caso de un fallo de conexión con la base de datos, se recuperaría indicando una reconexión a esta.	
Observaciones N/A	
Prototipo de interfaz N/A	

2.4.3. Requisitos no funcionales

Los requisitos no funcionales son aquellos que definen las restricciones del sistema, son propiedades, cualidades que el sistema debe poseer. Representan las características del producto [Sommerville(2005)].

En cuanto al sistema a desarrollar, cuenta con los siguientes requisitos:

Fiabilidad: Debe ser capaz de recuperarse ante fallos como errores de conexión con la base de datos, errores de configuración en el tamaño de almacenamiento de datos temporales.

Soporte: El diseño tendrá en cuenta patrones de la Banda de los Cuatro (GOF, por sus siglas en inglés) y Patrones Generales de Software para Asignar Responsabilidades (GRASP, por sus siglas en inglés) para garantizar la escalabilidad del sistema.

Seguridad: La comunicación externa con el servidor de base de datos de histórico se realiza a través de Capa Segura de Comunicaciones (SSL, por sus siglas en inglés).

Legales: Las herramientas y componentes empleados para el desarrollo de la solución deben ser libres y permitir incluirse dentro de *software* privado.

2.5. Propuesta de solución

Luego de haber expuesto la metodología de desarrollo utilizada, las herramientas y tecnologías, describir el dominio e identificar los requisitos a desarrollar; se puede definir la propuesta de solución:

La biblioteca de acceso a datos, al recibir una consulta de un intervalo de puntos o logs, se conecta, a través de la capa de comunicaciones, al servidor de históricos, para obtener los datos de

la base de datos donde estos se encuentran alojados. Se realiza un análisis, con los datos que se encuentran almacenados temporalmente dentro de su sistema de caché, tanto en la RAM como en el HDD. Es llevado a cabo una división de pedidos para determinar la fuente de los datos:

Caché RAM: Encargado de obtener los elementos alojados en la caché RAM, determinado por el proceso de búsqueda de elementos.

Caché HDD: Encargado de obtener los elementos alojados en la caché HDD, determinado por el proceso de búsqueda de elementos.

Servidor de base de datos: Encargado de obtener los elementos que no se encuentran en la caché, realizando una conexión directamente con el servidor de base de datos.

Por último se obtienen cada vez, por cada fuente, los datos del pedido, devolviéndolos a la aplicación cliente ordenados por *timestamp*. Todos los elementos provenientes de la base de datos, se intentarán almacenar en la caché. En cuanto se llegue al límite, almacenando los datos en la caché de RAM, se selecciona un candidato (haciendo uso del algoritmo de reemplazo configurado) y se inserta en la caché del HDD. Al igual que ocurre con la caché de RAM, en cuanto se llene la del HDD, el datos seleccionado será eliminado definitivamente.

2.6. Arquitectura de la libHDA

Pressman [[Pressman\(2010\)](#)] plantea que “(...) *la arquitectura de software es la estructura u organización de los componentes del programa (módulos), la manera en que estos componentes interactúan, y la estructura de datos que utilizan los componentes (...)*”.

La arquitectura brinda una visión global del sistema. De esta manera, se entiende, organiza su desarrollo, plantea la reutilización de *software* y permite hacerlo evolucionar. Exige diseñar muy cuidadosamente la arquitectura bajo la cual funciona el sistema, ya que las decisiones que se tomen tendrán una elevada repercusión a lo largo de todo el ciclo de desarrollo de la solución. Para definirla es necesario seleccionar y combinar patrones [[Reynoso\(2004\)](#)].

Los patrones de diseño pueden usarse durante el diseño del *software*. Una vez que se halla desarrollado el modelo de análisis, el diseñador puede examinar una representación detallada del problema que debe resolver y las restricciones que impone el problema. La descripción del problema se examina en varios grados de abstracción para determinar si es flexible para uno o más de los siguientes tipos de patrones de diseño [[Pressman\(2010\)](#)].

Patrones arquitectónicos: Estos patrones definen la estructura general del *software*, indican las relaciones entre los subsistemas y los componentes del *software* y definen las reglas para especificar las relaciones entre los elementos (clases, paquetes, componentes, subsistemas) de la arquitectura.

Patrones de diseño: Estos patrones se aplican en un elemento específico del diseño como un agregado de componentes o los mecanismos para efectuar la comunicación de componente a componente.

Idiomas: A veces llamados “*patrones de código*”, estos patrones específicos del lenguaje por lo general implementan un elemento algorítmico o un componente, un protocolo de interfaz específico o un mecanismo de comunicación entre los componentes.

2.6.1. Estilo y patrones arquitectónicos

Para la solución propuesta se hace uso de la **arquitectura centrada en datos** debido a que el centro de esta arquitectura reside en un lugar donde se almacenan los datos (base de datos donde el servidor de históricos almacena todos los puntos, logs, estados de conexión, etc.) [[Pressman\(2010\)](#)].

2.6.2. Patrones de diseño

Un patrón de diseño constituye una solución estándar para un problema común de programación en el desarrollo del *software*. Además de ser una técnica muy eficaz para flexibilizar el código haciéndolo satisfacer ciertos criterios, así como permitir de una manera más práctica describir ciertos aspectos de la organización de un programa [[Gamma\(2006\)](#)].

Para el desarrollo de la solución se tuvieron en cuenta los GRASP, describiendo principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones [[Larman\(2003\)](#)].

Creador: Es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado [[Larman\(2003\)](#)]. Un ejemplo de este patrón se evidencia en la clase `CacheSystemPoint`.

Controlador: Asigna la responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase [[Larman\(2003\)](#)]. Un ejemplo de este patrón se evidencia en la clase `DataAccessPrivate`.

Polimorfismo: La responsabilidad de definir la variación de comportamiento basado en tipos se asigna a aquellos para los cuales esta variación ocurre. Esto se logra mediante el uso de operaciones polimórficas [[Larman\(2003\)](#)]. En `AbstractQuery` se evidencia su uso.

Además de los patrones anteriores, se utilizaron los GOF, los cuales se clasifican en dependencia del propósito para los que hayan sido definidos: de creación, estructurales y de comportamiento [[Guerrero and Suárez\(2013\)](#)].

Solitario/Singleton: Como comenta Steve Holzner [Holzner(2006)], este patrón se asegura de que se pueda instanciar solamente un objeto de una clase en particular. En caso de no usar este patrón, se crearían objetos cada vez que se requiera una instancia de la clase, realizando un consumo más elevado de la memoria RAM, sin tener en cuenta que los objetos no comparten un estado intrínseco con la clase que lo llama.

2.7. Modelo de diseño

Un modelo de diseño es un modelo de objetos que se centra en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tiene impacto en el sistema a considerar. Además, el modelo de diseño sirve de absorción de la implementación del sistema, y es de ese modo, utilizada como una entrada fundamental de las actividades de implementación [Jacobson and Booch(2000)].

2.7.1. Diagramas de Clases del Diseño

Los diagramas de clases exponen un conjunto de interfaces, clases, con sus colaboraciones y restricciones. Se utilizan para modelar la vista de diseño estática de un sistema. Son importantes para visualizar, especificar, documentar modelos estructurales y construir sistemas ejecutables aplicando ingeniería directa e inversa [Tamayo and Pulupa(2013)]. En el diagrama que se muestra a continuación se relacionan todas las clases de la solución. Debido a la cantidad de atributos y operaciones que contienen, se determinó dejar solo el nombre de la clase. En la sección siguiente se describen detenidamente cada una de estas.

2.7.2. Descripción de las Clases del Diseño

Se describen a continuación las clases presentes en la Figura 2.2.

Tabla 2.18. Descripción de CacheSystem.

Descripción de la clase CacheSystem	
Nombre CacheSystem	
Tipo de clase Controladora	
Atributos	Tipos
_conn	PG::Connection
_cacheTypeRAM	CacheAlgorithm::RAM::Type
_cacheTypeHDD	CacheAlgorithm::HDD::Type
Continúa en la próxima página	

Tabla 2.18 – Continuada desde la página anterior

Descripción de la clase CacheSystem	
_dataSize	int
_memStorageLimit	uint64_t
_memTimeLimit	int
_diskStorageLimit	uint64_t
_diskTimeLimit	int
_diskStorageDir	boost::filesystem::path
_memCacheSize	uint64_t
_lockMutex	boost::mutex
_recycleFunctionsRAM	RecycleFunction[CacheAlgorithm::RAM::ALGORITHMS_SIZE]
_recycleFunctionsHDD	RecycleFunction[CacheAlgorithm::HDD::ALGORITHMS_SIZE]
Responsabilidades	
Nombre	cacheRecyclingTypeRAM() const: CacheAlgorithm::RAM::Type
Descripción	Retorna el tipo de reciclado que se realiza cuando la caché de la RAM llega al límite de almacenamiento.
Nombre	cacheRecyclingTypeHDD() const: CacheAlgorithm::HDD::Type
Descripción	Retorna el tipo de reciclado que se realiza cuando la caché del HDD llega al límite de almacenamiento.
Nombre	setCacheRecyclingTypeRAM(const CacheAlgorithm::RAM::Type): void
Descripción	Define el tipo de reciclado a utilizar para la RAM.
Nombre	setCacheRecyclingTypeHDD(const CacheAlgorithm::HDD::Type): void
Descripción	Define el tipo de reciclado a utilizar para el HDD.
Nombre	setConnection (const PG::Connection &): void
Descripción	Es utilizado para establecer la conexión con el servidor de base de datos, realizando un clonado, para tener exclusividad.
Nombre	setMemStorageLimit(const uint32_t): void
Descripción	Método para establecer el límite en cuanto a tamaño de almacenamiento de elementos en RAM antes de que se comience el proceso de reciclado de datos utilizando el algoritmo establecido.
Nombre	setMemTimeLimit(const int): void
Descripción	Realiza una limpieza en la caché cada cierto tiempo (expresado en minutos), borrando todos los datos de la memoria RAM. Para deshabilitar se debe pasar un 0.
Nombre	setDiskStorageLimit(const uint32_t): void
Continúa en la próxima página	

Tabla 2.18 – Continuada desde la página anterior

Descripción de la clase CacheSystem	
Descripción	Método para establecer el límite en cuanto a tamaño de almacenamiento de elementos en HDD antes de que se comience el proceso de reciclado de datos utilizando el algoritmo establecido.
Nombre	setDiskStorageDir(const std::string &): void
Descripción	Establece el directorio donde se almacenará la caché en HDD.
Nombre	clear(): virtual void
Descripción	Elimina toda los datos de la caché.

Tabla 2.19. Descripción de CacheSystemPoint.

Descripción de la clase CacheSystemPoint	
Nombre	CacheSystemPoint
Tipo de clase	Controladora
Atributos	Tipos
_pointSize	int
_tsData	PointDataTimeStamp
_using	std::map<TimeStamp, bool>
_tsRelation	std::map<TimeStamp, TimeStamp>
_tsAccessTime	std::map<TimeStamp, uint64_t>
_tsAccessCount	std::map<TimeStamp, uint64_t>
_apLimit	uint64_t
_dpLimit	uint64_t
_lastTs	TimeStamp
_lastTsData	PointDataMap *
_lastTsUsing	bool *
_lastId	uint32_t
_lastIdVector	PointVector *
Responsabilidades	
Nombre	static instance(): CacheSystemPoint *
Descripción	Devuelve la instancia única del caché de puntos.
Nombre	add(const Point &, TimeStamp): bool
Descripción	Almacena en la caché un punto referente a un rango de tiempo.
Continúa en la próxima página	

Tabla 2.19 – Continuada desde la página anterior

Descripción de la clase CacheSystemPoint	
Nombre	<code>find(const CacheIdSet &, const CacheIdSet &, const uint64_t, const uint64_t, PAbstractQueryPointVector &): void</code>
Descripción	Realiza la búsqueda en la caché de un identificador específico comprendido entre dos estampas de tiempo.
Nombre	<code>clear(): void</code>
Descripción	Elimina todos los puntos que no se estén utilizando de la caché.
Nombre	<code>endInsert(): void</code>
Descripción	Indica a la caché de que se terminó una sesión de pedido (insertado) de puntos.

Tabla 2.20. Descripción de CacheSystemLog.

Descripción de la clase CacheSystemLog	
Nombre CacheSystemLog	
Tipo de clase Controladora	
Atributos	Tipos
<code>_logSize</code>	<code>int</code>
<code>_tsData</code>	<code>LogTimeStampMap</code>
<code>_using</code>	<code>std::map<TimeStamp, bool></code>
<code>_tsRelation</code>	<code>std::map<TimeStamp, TimeStamp></code>
<code>_tsAccessTime</code>	<code>std::map<TimeStamp, uint64_t></code>
<code>_tsAccessCount</code>	<code>std::map<TimeStamp, uint64_t></code>
<code>_logLimit</code>	<code>uint64_t</code>
Responsabilidades	
Nombre	<code>static instance(): CacheSystemPoint *</code>
Descripción	Devuelve la instancia única del caché de Log.
Nombre	<code>add(const Point &, TimeStamp, const LogFilterCriteria &): bool</code>
Descripción	Almacena en la caché un Log referente a un rango de tiempo y un filtro.
Nombre	<code>find(const CacheIdSet &, const uint64_t, const uint64_t, const LogFilterCriteria &, const long, PAbstractQueryLogVector &): void</code>
Descripción	Realiza la búsqueda en la caché de un identificador específico comprendido entre dos estampas de tiempo.
Nombre	<code>clear(): void</code>
Continúa en la próxima página	

Tabla 2.20 – Continuada desde la página anterior

Descripción de la clase CacheSystemLog	
Descripción	Elimina todos los Log que no se estén utilizando de la caché.
Nombre	endInsert(): void
Descripción	Indica a la caché de que se terminó una sesión de pedido (insertado) de Log.

Tabla 2.21. Descripción de AbstractQuery<Type>.

Descripción de la clase AbstractQuery<Type>	
Nombre	AbstractQuery<Type>
Tipo de clase	Abstracta
Atributos	Tipos
_data	Type
_range	TimeStamp
_dataSource	DataSource::Type
Responsabilidades	
Nombre	AbstractQuery(const DataSource::Type)
Descripción	Constructor por defecto.
Nombre	virtual AbstractQuery()
Descripción	Destructor virtual por defecto.
Nombre	range() const: const TimeStamp &
Descripción	Devuelve el rango del pedido.
Nombre	dataSource() const: DataSource::Type
Descripción	Retorna el origen de los datos.
Nombre	virtual get() = 0: const Type &
Descripción	A ser implementado por cada hijo de AbstractQuery, encargados de obtener los datos de una fuente determinada y devolverlo tupla por tupla, por cada llamada a esta función.

Tabla 2.22. Descripción de CacheQuery<Type>.

Descripción de la clase CacheQuery<Type>	
Nombre	CacheQuery<Type>
Continúa en la próxima página	

Tabla 2.22 – Continuada desde la página anterior

Descripción de la clase CacheQuery<Type>	
Tipo de clase Controladora	
Atributos	Tipos
_using	bool *
_end	bool
_it	std::vector<Type>::const_iterator
_limit	std::vector<Type>::const_iterator
Responsabilidades	
Nombre	CacheQuery (bool *, const TimeStamp &, std::vector<Type>::const_iterator, std::vector<Type>::const_iterator)
Descripción	Constructor por defecto.
Nombre	CacheQuery()
Descripción	Destructor por defecto.
Nombre	get(): const Type &
Descripción	Devuelve el dato en cada llamada, aumentando el iterador interno.

Tabla 2.23. Descripción de DiskQueryLog.

Descripción de la clase DiskQueryLog	
Nombre DiskQueryLog	
Tipo de clase Controladora	
Atributos	Tipos
_readBuffer	static const int
_dataFile	boost::filesystem::ifstream
_tsDir	boost::filesystem::path
_unpacker	msgpack::unpacker
_counter	uint64_t
Responsabilidades	
Nombre	DiskQueryLog(const boost::filesystem::path &, const TimeStamp &, uint32_t, const LogFilterCriteria &)
Descripción	Constructor por defecto.
Nombre	DiskQueryLog()
Descripción	Destructor por defecto.
Continúa en la próxima página	

Tabla 2.23 – Continuada desde la página anterior

Descripción de la clase <code>DiskQueryLog</code>	
Nombre	<code>get(): const Log &</code>
Descripción	Devuelve un <code>Log</code> por cada llamada realizada, ordenados por <code>timestamp</code> .

Tabla 2.24. Descripción de `DiskQueryPoint`.

Descripción de la clase <code>DiskQueryPoint</code>	
Nombre	<code>DiskQueryPoint</code>
Tipo de clase	Controladora
Atributos	Tipos
<code>_readBuffer</code>	<code>static const int</code>
<code>_dataFile</code>	<code>boost::filesystem::ifstream</code>
<code>_tsDir</code>	<code>boost::filesystem::path</code>
<code>_unpacker</code>	<code>msgpack::unpacker</code>
<code>_counter</code>	<code>uint64_t</code>
Responsabilidades	
Nombre	<code>DiskQueryPoint(const boost::filesystem::path &, const TimeStamp &, uint32_t)</code>
Descripción	Constructor por defecto.
Nombre	<code>DiskQueryPoint()</code>
Descripción	Destructor por defecto.
Nombre	<code>get(): const Point &</code>
Descripción	Devuelve un <code>Point</code> por cada llamada realizada, ordenados por <code>timestamp</code> .

2.8. Modelo de Despliegue

Este modelo (Figura 2.3) establece una correspondencia entre la arquitectura de *software* y la arquitectura de *hardware* del sistema. Muestra la configuración de los nodos de procesamiento en tiempo de ejecución, los enlaces de comunicación entre ellos, y las instancias de los componentes y objetos que residen en ellos. El modelo de despliegue se utiliza para capturar los elementos de configuración del procesamiento y las conexiones entre esos elementos. También se utiliza para visualizar los nodos procesadores la distribución de los procesos y los componentes [Larman(2003)].

las bases para la futura realización de las características con la que debe contar la solución propuesta. A través del estudio de los estilos y patrones arquitectónicos, se seleccionaron los más adecuados a la solución, lo que rindió como base para la definición del diseño de clases. Teniendo en cuenta las historias de usuario se aprecia con mayor detalle cada una de las características a implementar.

CAPÍTULO 3

IMPLEMENTACIÓN Y PRUEBAS

3.1. Introducción

En el presente capítulo se comentará acerca del proceso de implementación del sistema partiendo de los modelos, diagramas, herramientas y métodos presentados en los capítulos anteriores. Además, se tendrá en cuenta el proceso de aplicación de pruebas unitarias de la solución, determinando el cumplimiento con los requisitos funcionales y no funcionales.

3.1.1. Selección de biblioteca de serializado

Una de las características que contendrá el sistema creado será la posibilidad de usar el HDD como almacenamiento adicional a la RAM para guardar los datos de la caché. El proceso de guardar dichos datos desde la RAM hacia el HDD pasa a través de la serialización de datos, que no es más que la representación en forma de bytes de cualquier objeto en cuestión, para luego ser almacenado o transportado. Cada lenguaje de programación tiene sus mecanismos internos para realizarlo. En el caso de C++, este no trae consigo un mecanismo intuitivo para dicho proceso, además de tener en cuenta muchos factores, como sistema operativo, tamaño de los tipos de datos, arquitectura, orden de los bytes en el almacenamiento, etc. En el presente proyecto, los datos almacenados en el HDD, no se utilizarán por elementos externos, de ahí que la mayoría de las preocupaciones a tener en cuenta en un sistema de serialización no se tienen en cuenta. Como se expuso en la Subsección 1.4.1 existen bibliotecas para agilizar el proceso en C++. Como se trata de devolver al cliente¹ los datos de la manera más rápida posible, se debe seleccionar la biblioteca que más rápido serializa, teniendo en cuenta el aditivo secundario de que el tamaño de

¹Se refiere a la aplicación que se encuentra utilizando la libHDA

los datos finales a serializar sea menor, pues a mayor distancia de relación

Cantidad de datos a guardar \implies Tamaño de fichero generado

menor tiempo en la lectura cuando se requiera recuperar dichos datos. En la Tabla 3.1 y la Tabla 3.2 se muestran las pruebas realizadas a tres de las bibliotecas mencionadas, tomando de referencia el tiempo en milisegundos y el tamaño en megabytes. La prueba se realizó utilizando un millón de datos con cien pasadas².

Tabla 3.1. Pruebas realizadas para determinar velocidad en milisegundos de la serialización.







Biblioteca	Tiempo de Serializado	
MsgPack 0.5.8	20.00 \pm 1.00	
Boost::Serialization 1.55.0.2	71.00 \pm 5.00	
Protobuff 1.0.2	25.00 \pm 3.00	

Tabla 3.2. Pruebas realizadas para determinar el tamaño de los datos en megabytes de la serialización.

Biblioteca	Tamaño de Serializado	
MsgPack 0.5.8	6.55	
Boost::Serialization 1.55.0.2	11.44	
Protobuff 1.0.2	7.61	

En las pruebas no se incluye s11n (versión 1.3.1) puesto que los resultados que arrojó fueron muy desfasados con respecto a los otros, numéricamente sería de un tiempo de 5963,5 milisegundos y en cuanto al tamaño del serializado de 47,1 megabytes. Hay que tener en cuenta la característica propia de s11n que no incluye formatos binarios para la serialización, de ahí esos resultados.

Todos las pruebas de rendimiento se realizaron en una PC con las siguientes características:

CPU: Intel® Core™i7-4702MQ @ 2.20GHz

RAM: Samsung® DDR3 8GB @ 1333Mhz

²Se refiere a la cantidad de veces que se probó cada serializador

Sistema Operativo: Debian 8.0

Arquitectura: amd64 (64 bits)

Núcleo: Linux 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt4-3

Compilador: GNU gcc 4.9.2

De los resultados anteriores se opta por MsgPack como biblioteca ha ser utilizada para la realización puesto que se presenta como la más rápida y la que menos tamaño de datos genera, además en cuanto a la velocidad es la más estable, teniendo una variación solamente de un milisegundo. Ésta última en comparación con Protobuf no necesita la generación de un fichero interfaz para luego generar un código fuente para el lenguaje a utilizar, donde en el caso particular de la solución propuesta no es necesario.

3.2. Modelo de implementación

Este modelo comprende un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes se encuentran datos, archivos, ejecutables, código fuente y los directorios. Se describe, de manera fundamental, la relación existente entre los paquetes y clases del modelo de diseño a subsistemas y componentes físicos [Jacobson and Booch(2000)] [Hernández(2013)]. Se conforma por:

- Las dependencias entre las partes de código del sistema (diagramas de componentes).
- La estructura del sistema en ejecución (diagrama de despliegue). Visto en Sección 2.8.

3.2.1. Diagramas de componentes

Diagrama representativo de la relación entre los componentes del sistema; dichos componentes representan una parte modular, desplegable y reemplazable, que encapsula la implementación y expone un conjunto de interfaces [Group(2001)]. Podría ser código fuente, un fichero binario o ejecutable, navegadores, servidores de Protocolo de Transferencia de Hipertexto (HTTP, por sus siglas en inglés) o SQL, entre otros [Larman(2003)]. Se observa en la Figura 3.1 el de la actual solución.

3.3. Modelo de Pruebas

Las pruebas de *software* son una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, cuyos resultados son observados, registrados y analizados. Se de-

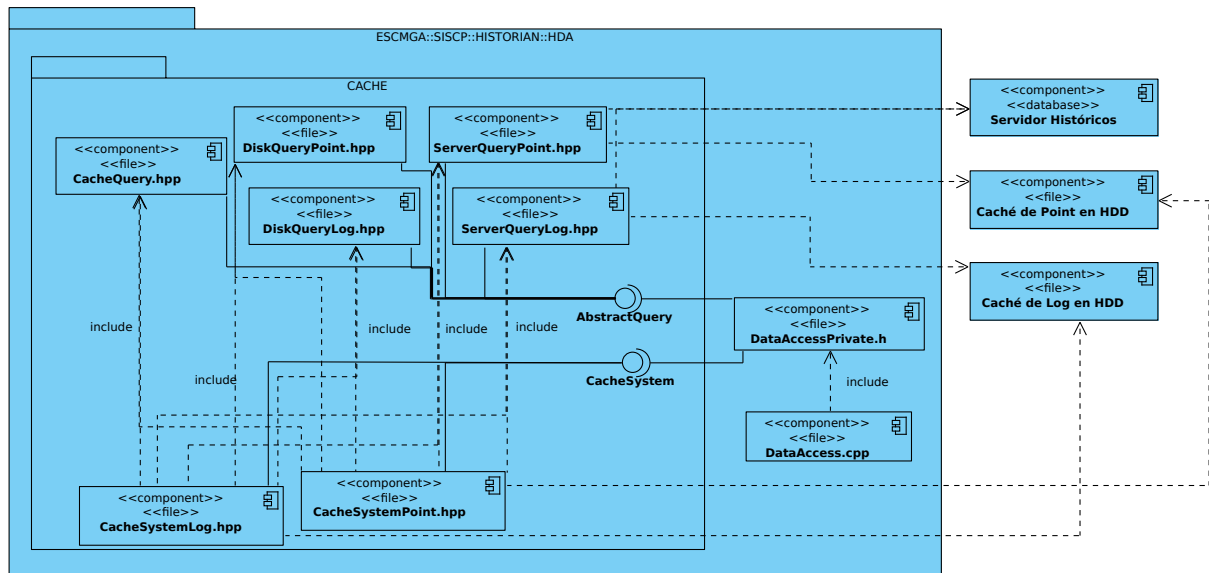


Figura 3.1. Diagrama de componentes

fine como el proceso de ejercitar o evaluar el sistema por medios manuales o automáticos, para verificar que satisface los requerimientos o, para identificar diferencias entre los resultados esperados y los que produce dicho sistema.

Para lograr el objetivo de las pruebas de *software* es necesario planear y ejecutar una serie de pasos (pruebas de unidad, integración, validación y sistema). Las pruebas de unidad e integración se concentran en la verificación funcional de cada componente y en la incorporación de componentes en la arquitectura del *software*. La prueba de validación demuestra el cumplimiento con los requisitos del *software* y la prueba del sistema valida el *software* una vez que se ha incorporado a un elemento superior.

Existen dos categorías diferentes (según [Pressman(2005)]) de técnicas de diseño de casos de prueba:

Pruebas de caja blanca: Se concentran en la estructura de control del programa. Los casos de prueba se derivan para asegurar que todas las instrucciones del programa se ejecuten por lo menos una vez durante la prueba, además de que todas las condiciones lógicas se ejerciten.

Pruebas de caja negra: Están diseñadas para validar requisitos funcionales sin importar el funcionamiento interno de un programa. Los casos de prueba se derivan mediante la partición de los dominios de entrada y salida de un programa, en forma tal que proporciones cobertura completa. Con respecto al sistema propuesto, no se realizan pruebas de caja negra.

Se establecen y ejecutan una serie de pruebas con el objetivo de validar el sistema desarrollado en cuanto a los objetivos antes mencionados, en cuyo caso se realizarán pruebas de caja

blanca. Con las pruebas se comprueba la eficiencia, corroborando que se ha cumplido el objetivo general del presente documento.

3.3.1. Diseño de casos de prueba

Los casos de pruebas son un conjunto de acciones de entradas que deben corresponderse con un estado de salida esperado. A continuación (Tabla 3.3) se enumeran los casos tenidos en cuenta para la comprobación de las funcionalidades del sistema, basándose en los requisitos funcionales del sistema. Se utiliza un conjunto de datos generados aleatoriamente para su uso y comprobaciones.

Tabla 3.3. Casos de prueba.

No	Caso de prueba	Resultado esperado
1	Se realiza un pedido de puntos analógicos.	Se almacena en RAM los datos devueltos por el servidor de históricos dependiendo de la configuración de espacio de almacenamiento y del estado actual del almacenamiento.
2	Se realiza un pedido de puntos digitales.	Se almacena en RAM los datos devueltos por el servidor de históricos dependiendo de la configuración de espacio de almacenamiento y del estado actual del almacenamiento.
3	Se realiza un pedido de eventos.	Se almacena en RAM los datos devueltos por el servidor de históricos dependiendo de la configuración de espacio de almacenamiento y del estado actual del almacenamiento.
4	Se realiza un pedido de puntos analógicos y no queda espacio en el almacenamiento de RAM.	Se escoge a través del algoritmo configurado el intervalo a eliminar y se almacena en el HDD.
5	Se realiza un pedido de puntos digitales y no queda espacio en el almacenamiento de RAM.	Se escoge a través del algoritmo configurado el intervalo a eliminar y se almacena en el HDD.
Continúa en la próxima página		

Tabla 3.3 – Continuada desde la página anterior

No	Caso de prueba	Resultado esperado
6	Se realiza un pedido de eventos y no queda espacio en el almacenamiento de RAM.	Se escoge a través del algoritmo configurado el intervalo a eliminar y se almacena en el HDD.
7	Se configura un tamaño límite determinado de almacenamiento en RAM y se realiza un pedido de un intervalo que implique un tamaño total mayor que el configurado.	El tamaño de los datos almacenados debe ser menor o igual al tamaño límite configurado.
8	Se configura un tamaño límite determinado de almacenamiento en HDD y se realiza un pedido de un intervalo que implique un reciclado de la caché de un tamaño de datos total mayor que el configurado.	El tamaño de los datos almacenados debe ser menor o igual al tamaño límite configurado.
9	Se configura el directorio donde se almacenarán los datos en el HDD y se realiza un pedido de un intervalo que implique un reciclado de la caché.	En el directorio deben quedar almacenados los datos exactos que fueron eliminados del almacenamiento en RAM.
10	Se realiza un pedido de puntos analógicos.	Los datos devueltos deben estar ordenados por <i>timestamp</i> .
11	Se realiza un pedido de puntos digitales.	Los datos devueltos deben estar ordenados por <i>timestamp</i> .
12	Se realiza un pedido de eventos.	Los datos devueltos deben estar ordenados por <i>timestamp</i> .
13	Se solicita una limpieza de la caché al controlador.	Todos los datos de la caché deben eliminarse, excluyendo los datos utilizados por otras consultas concurrentes.
14	Se deshabilita la caché en RAM y se realiza un pedido de puntos analógicos.	No se realiza ninguna operación sobre el almacenamiento en RAM.
15	Se deshabilita la caché en RAM y se realiza un pedido de puntos digitales.	No se realiza ninguna operación sobre el almacenamiento en RAM.
16	Se deshabilita la caché en RAM y se realiza un pedido de eventos.	No se realiza ninguna operación sobre el almacenamiento en RAM.

Continúa en la próxima página

Tabla 3.3 – Continuada desde la página anterior

No	Caso de prueba	Resultado esperado
17	Se deshabilita la caché en HDD y se realiza un pedido de puntos analógicos.	No se realiza ninguna operación sobre el almacenamiento en HDD.
18	Se deshabilita la caché en HDD y se realiza un pedido de puntos digitales.	No se realiza ninguna operación sobre el almacenamiento en HDD.
19	Se deshabilita la caché en HDD y se realiza un pedido de eventos.	No se realiza ninguna operación sobre el almacenamiento en HDD.

Se tuvo en cuenta además las pruebas unitarias a cada clase implementada, adicionando un nivel de seguridad adicional informando acerca del correcto funcionamiento de cada una de las clases. A continuación se presentan un ejemplo de esos casos en la Tabla 3.4, conjuntamente con las tareas realizadas.

Tabla 3.4. Casos de prueba unitarias.

Caso	Nombre
1	CacheQueryPointTest
Prueba	<i>Descripción</i>
Case1	Se comprueba que indicando un intervalo de un vector de puntos lo devuelve en el mismo orden. Cuando llega al límite, establece la variable de uso en falso y devuelve un punto nulo.
Case2	Se comprueba para el caso de llamadas repetidas al método de obtención luego de haber llegado al límite, en cuyo caso devuelve un punto nulo.
Case3	Se comprueba para el caso de eliminación antes de haber devuelto todos los puntos, donde se debe establecer la variable de uso en falso.
Caso	Nombre
2	CacheQueryLogTest
Prueba	<i>Descripción</i>
Case1	Se comprueba que indicando un intervalo de un vector de eventos lo devuelve en el mismo orden. Cuando llega al límite, establece la variable de uso en falso y devuelve un evento nulo.
Case2	Se comprueba para el caso de llamadas repetidas al método de obtención luego de haber llegado al límite, en cuyo caso devuelve un evento nulo.
Continúa en la próxima página	

Tabla 3.4 – Continuada desde la página anterior

Case3	Se comprueba para el caso de eliminación antes de haber devuelto todos los eventos, donde se debe establecer la variable de uso en falso.
-------	---

3.3.2. Framework para pruebas “*Google C++ Testing Framework*”

Framework de Google para la programación de pruebas en C++ en una variedad de plataformas. Basado en la arquitectura xUnit. Soporta descubrimiento automatizado de pruebas, un gran conjunto de afirmaciones³, afirmaciones definidas por el usuario, pruebas muertas, fallas fatales y no fatales, varias opciones para ejecutar las pruebas y generación de reportes de pruebas en XML [Google(2014c)].

Google Test está diseñado para tener requerimientos medianamente mínimos para construirse y usarse con sus proyectos. Actualmente soporta GNU/Linux, Microsoft® Windows™, Apple® MacOSX® y Cygwin [Google(2014c)].

Este framework se utiliza para la realización de las pruebas de caja blanca en la solución propuesta debido a la poca cantidad de código necesaria para realizar alguna prueba, así como su eficiencia y los reportes de tiempo que genera. Las pruebas se realizaron teniendo relación con las clases que modelan el negocio, para permitir el trabajo con la estructura interna de dichas clases y de esta manera tener mayor control de las afirmaciones.

Resultados de las pruebas realizadas

Se presenta en la Figura 3.2 un ejemplo de salida de la ejecución de las pruebas, además de una captura de pantalla de la página web generada a partir de los datos recogidos para la cobertura del código fuente (Figura 3.3).

Como resultado de las pruebas, se detectaron 6 no conformidades, en cuanto a los resultados esperados, centradas en las clases `TimeStamp`, `ServerQueryPoint`, `CacheSystemPoint` y `CacheSystemLog`. Todas fueron resueltas satisfactoriamente, logrando que el resultado de las pruebas fuera de un 100% de aceptación.

3.3.3. Validación de la propuesta a través de la eficiencia y eficacia

Dichos términos, aunque parecidos no se deben confundir, se argumenta su significado sobre el impacto del presente trabajo en la inclusión de la solución a la biblioteca de acceso a datos del SCADA Galba, según [Española(2015)]:

³Se refiere a los métodos que hacen que las pruebas fallen dependiendo de las comparaciones realizadas.

```

Running main() from gtest_main.cc
[====] Running 24 tests from 6 test cases.
[-----] Global test environment set-up.
[-----] 3 tests from CacheQueryPointTest
[ RUN ] CacheQueryPointTest.Case1
[ OK ] CacheQueryPointTest.Case1 (1 ms)
[ RUN ] CacheQueryPointTest.Case2
[ OK ] CacheQueryPointTest.Case2 (0 ms)
[ RUN ] CacheQueryPointTest.Case3
[ OK ] CacheQueryPointTest.Case3
[-----] 8 tests from TimeStampTest
[-----] 3 tests
[ RUN ] TimeStampTest.DefaultConstructor
[ OK ] TimeStampTest.DefaultConstructor (0 ms)
[ RUN ] TimeStampTest.DefaultConstructorWithValues
[ OK ] TimeStampTest.DefaultConstructorWithValues (0 ms)
[ RUN ] TimeStampTest.DefaultConstructorWithString
[ OK ] TimeStampTest.DefaultConstructorWithString (1 ms)
[ WARN ] TimeStamp: Cadena inválida para convertir: "000254error-555478"
[ WARN ] TimeStamp: Cadena inválida para convertir: "000254error555478"
[ WARN ] TimeStamp: Cadena inválida para convertir: "-0002545-55478"
[ OK ] TimeStampTest.DefaultConstructorWithString (1 ms)
[-----] 3 tests
[ RUN ] TimeStampTest.CompareRanges
[ OK ] TimeStampTest.CompareRanges (0 ms)
[ RUN ] TimeStampTest.ConvertToString
[ OK ] TimeStampTest.ConvertToString (0 ms)
[ RUN ] TimeStampTest.ConvertToDirString
[ OK ] TimeStampTest.ConvertToDirString (0 ms)
[ RUN ] TimeStampTest.Equality
[ OK ] TimeStampTest.Equality (0 ms)
[ RUN ] TimeStampTest.Inequality
[ OK ] TimeStampTest.Inequality (0 ms)
[-----] 8 tests from TimeStampTest (1 ms total)
[-----] Global test environment tear-down
[====] 24 tests from 6 test cases ran. (13858 ms total)
[ PASSED ] 24 tests.
    
```

Figura 3.2. Salida acotada de las pruebas con el framework Google Test

LCOV - code coverage report

Current view: [top level](#) - [src/cache/impl](#) - [AbstractQuery.cpp](#) (source / functions)

Test:	Hit	Total	Coverage
unit_test.info.cleaned	7	7	100.0 %
Date: 2015-05-19 22:14:27	Lines: 6	6	100.0 %
Legend: Lines: Hit: █ Miss: █	Functions: 6	6	100.0 %

```

Line data   Source code
1           /*
2           * Created on: 27 January, 2015
3           * Author: Luis Felipe Dominguez Vega <lfdominguezestudiantes.uci.cu>
4           *
5           *
6           *
7           *
8           *
9           *
10          */
11
12          #include <set>
13
14          #include "historian/hda/Global.hpp"
15          #include "historian/hda/Variant.h"
16
17          HDA_BEGIN_NAMESPACE
18          namespace CACHE
19          {
20          {
21
22          template<class Type>
23          AbstractQueryType::AbstractQuery(const DataSource::Type dataSource) :
24          dataSource(dataSource)
25          {
26          }
27
28          template<class Type>
29          DataSource::Type AbstractQueryType::dataSource() const
30          {
31          return dataSource;
32          }
33
34          template<class Type>
35          const TimeStamp &AbstractQueryType::range() const
36          {
37          return range;
38          }
39          }
40
41          HDA_END_NAMESPACE
42
    
```

Generated by: [LCOV version 1.11](#)

Figura 3.3. Salida para la clase AbstractQuery del código cubierto por las pruebas

Eficiencia: Capacidad de disponer de alguien o de algo para conseguir un efecto determinado. Obtener mejores resultados con menos recursos.

Eficacia: Capacidad de lograr el efecto que se desea o se espera. Tener resultados con los recursos que necesita.

En cuanto a la eficiencia y tomando en cuenta la presente solución, se afirma que se ha optimizado el tiempo de obtención de datos que han sido pedidos con anterioridad. Dichas pruebas

son realizadas utilizando la Base de Datos Objeto-Relacional (ORDBMS, por sus siglas en inglés) PostgreSQL en su versión 9.4, conectándose a través de *sockets Unix*[®] en la PC local.

Tabla 3.5. Prueba realizada sin resultados de puntos.



Caché de Puntos	Tiempo de procesamiento (ms)	
No	20	
Si	13	

Tabla 3.6. Prueba realizada para 197'957 puntos.



Caché de Puntos	Tiempo de procesamiento (ms)	
No	559	
Si	121	

Tabla 3.7. Prueba realizada para 395'906 puntos.



Caché de Puntos	Tiempo de procesamiento (ms)	
No	1064	
Si	251	

Tabla 3.8. Prueba realizada para 494'740 puntos.



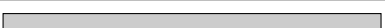

Caché de Puntos	Tiempo de procesamiento (ms)	
No	1388	
Si	314	

Tabla 3.9. Prueba realizada para 538'280 puntos.

Caché de Puntos	Tiempo de procesamiento (ms)	
No	1554	
Si	369	

Se hace uso racional de los medios disponibles, en este caso el computador donde se encuentra utilizándose la biblioteca de acceso a datos con el soporte de caché propuesto por la presente

tesis.

Para las pruebas de rendimiento, en cuanto a tiempo de respuesta del procesamiento de datos se utilizó una capacidad de cómputo igual a la empleada para la prueba de bibliotecas de serializado (Sección 3.1.1). Se ve reflejada la eficacia desde la Tabla 3.5 hasta la Tabla 3.9, donde se promedia una disminución $\approx 68,67\%$ en cuanto al tiempo requerido para la devolución de los datos. Se debe tener en cuenta que la convergencia en las pruebas fue de un 100% .

Para mostrar el comportamiento del mecanismo de caché, variando la cantidad de datos que convergen entre dos pedidos, se realizaron pruebas adicionales (Tabla 3.10 - Tabla 3.13). A diferencia de las pruebas anteriores, para aumentar la diversidad del entorno, en este caso se conecta a una ORDBMS externa (PostgreSQL 9.4).

Tabla 3.10. Prueba realizada para un 25% de convergencia de datos. Logrando un $8,19\%$ de optimización



Caché de Puntos	Tiempo de procesamiento (ms)	
No	517	
Si	480	

Tabla 3.11. Prueba realizada para un 50% de convergencia de datos. Logrando un $44,97\%$ de optimización



Caché de Puntos	Tiempo de procesamiento (ms)	
No	2011	
Si	1100	

Tabla 3.12. Prueba realizada para un 75% de convergencia de datos. Logrando un $78,84\%$ de optimización





Caché de Puntos	Tiempo de procesamiento (ms)	
No	2592	
Si	543	

Tabla 3.13. Prueba realizada para un 100% de convergencia de datos. Logrando un $86,86\%$ de optimización

Caché de Puntos	Tiempo de procesamiento (ms)	
No	3076	
Si	431	

3.4. Consideraciones del capítulo

De manera general acerca de la implementación del sistema, se tuvo en constante análisis el costo de cada operación realizada, ya que se intenta minimizar el tiempo que debe esperar la aplicación cliente entre un pedido y su respuesta. Se tuvo en cuenta la documentación del código fuente, haciendo uso de Doxygen, que facilite la rápida familiarización a futuros desarrolladores que quieran aportar o estudiar el mecanismo y la biblioteca en general; así como la elaboración con herramientas especializadas del informe de pruebas de cobertura, las cuales dan seguridad de que el código implementado por el autor ha sido probado en su totalidad; permitiendo que en futuras modificaciones no se incurra en regresiones. En cuanto a las pruebas realizadas, se corrigieron todas las no conformidades encontradas, así como en las pruebas de aceptación donde se observa, a partir de un 50 % de convergencia de datos, una optimización superior al 40 % del tiempo requerido para realizar las consultas a la base de datos de históricos.

CONCLUSIONES

La investigación realizada permitió arribar a las siguientes conclusiones:

1. El estudio de las herramientas y tecnologías actuales, permitió obtener el basamento teórico y tecnológico vinculado a los mecanismos de caché en sistemas SCADA.
2. La utilización de la metodología AUP (variación de la UCI) sentó las bases para estructurar, planificar, documentar y guiar el proceso de construcción del *software*, hasta su feliz término.
3. El análisis del mecanismo de caché, a la par de sus algoritmos, permitió establecer los requerimientos que sustentaron el desarrollo de la aplicación.
4. El análisis del diseño de la libHDA, para las modificaciones en la inclusión de las clases que dan solución al objeto de investigación, permitió comprobar que estos cambios no influyen en la implementación de las aplicaciones que hacen uso de la biblioteca.
5. Las pruebas unitarias, de cobertura y de aceptación permitieron validar la estabilidad y efectividad de la aplicación implementada.
6. El uso de un mecanismo de caché en la biblioteca de acceso a datos que almacena los elementos de las consultas, temporalmente, tanto en RAM como en HDD, logra disminuir el tiempo de respuesta, en los pedidos de datos históricos con rangos de tiempo solapados.
7. El uso de herramientas y bibliotecas con licencias libres, constituyen una fortaleza, que ofrece la posibilidad de su inclusión en software privado.
8. Al dotar la implementación con documentación del código fuente, facilita la rápida familiarización a futuros desarrolladores que quieran aportar o estudiar el mecanismo y la biblioteca.

RECOMENDACIONES

1. Para evitar sobrecarga de procesamiento en el sistema se recomienda que, cuando los pedidos a la base de datos no contienen áreas de convergencia, sea deshabilitado el mecanismo de caché.
2. Se le propone a la UCI que comparta las experiencias de esta investigación, con el proyecto en desarrollo del centro CEDIN, SCADA SAINUX.

ACRÓNIMOS

- AM** Modelado Ágil (*Agile Modeling*). 18
- ANSI** Instituto Nacional Estadounidense de Estándares (*American National Standards Institute*). 11, 22
- API** Interfaz de Programación de Aplicaciones (*Application Programming Interface*). 11
- ARC** Reemplazo de Caché Auto Configurable y de Baja Sobrecarga (*A self-tuning, low overhead Replacement Cache*). 17
- AUP** Proceso Unificado Ágil (*Agile Unified Process*). 18–20, 25
- BPMN** Modelo y Notación de Procesos de Negocio (*Business Process Model Notation*). 21
- CASE** Ingeniería de Software Asistida por Computadora (*Computer-Aided Software Engineering*). 21
- CEDIN** Centro de Informática Industrial. 1, 60
- Galba** Guardián del ALBA. 1, 2, 18, 20, 54
- GNU** GNU No es Unix (*GNU is Not Unix*). 14
- GOF** Banda de los Cuatro (*Gang-Of-Four*). 35, 37
- GRASP** Patrones Generales de Software para Asignar Responsabilidades (*General Responsibility Assignment Software Patterns*). 35, 37
- HDD** Unidad de Disco Duro (*Hard Disk Drive*). 9–11, 25, 27, 29–33, 36, 47, 51–53, 59

- HMI** Interfaz Hombre-Máquina (*Human-Machine Interface*). 1, 5
- HTML** Language de Marcado de Hipertexto (*HyperText Markup Language*). 19, 20
- HTTP** Protocolo de Transferencia de Hipertexto (*HyperText Transfer Protocol*). 49
- IBM** International Business Machines Corporation. 18
- IDE** Entorno Integrado de Desarrollo (*Integrated Development Enviroment*). 22–24
- ISO** Organización Internacional para la Normalización (*International Organization for Standardization*). 14, 22
- LFU** Menos Frecuentemente Usado (*Least Frequently Used*). 17
- libHDA** Biblioteca de Acceso a Datos de Históricos (*Historical Data Access library*). 1, 2, 4, 5, 10, 20, 24, 25, 28, 29, 33, 34, 36, 47, 59
- LRU** Menos Recientemente Usado (*Least Recently Used*). 16, 17
- MFC** Microsoft® Foundation Class. 14
- MRU** Más Recientemente Usado (*Most Recently Used*). 16, 17
- ODBMS** Administrador de Base de Datos Orientadas a Objetos (*Object-Oriented Database Management System*). 10
- ORDBMS** Base de Datos Objeto-Relacional (*Object-Relational DataBase Management System*). 56, 57
- POD** Tipos de Datos Planos (*Plain Old Data*). 13
- RAM** Memoria de Acceso Aleatorio (*Random Access Memory*). 6, 10, 11, 25, 27–31, 33, 36, 47, 51, 52, 59, 63
- RDBMS** Sistema de Administración de Base de Datos Relacional (*Relational DataBase Management System*). 22
- RDSMS** Sistema de Administración de Flujos de Datos Relacional (*Relational Data Stream Management System*). 22
- RR** Remplazo Aleatorio (*Random Replacement*). 17
- RTTI** Información de Tipos en Ejecución (*Runtime Types Information*). 11

- RUP** Proceso Unificado de Rational (*Rational Unified Process*). 18–20
- SCADA** Supervisión, Control y Adquisición de Datos (*Supervisory Control and Data Acquisition*). 1–3, 18, 20, 22, 54, 59, 60
- SQL** Lenguaje de Consulta Estructurado (*Structured Query Language*). 22, 49
- SRAM** RAM Estática (*Static RAM*). 6
- SSL** Capa Segura de Comunicaciones (*Secure Sockets Layer*). 35
- SysML** Lenguaje de Modelado de Sistemas (*System Modeling Language*). 21
- TDD** Desarrollo Guiado por Pruebas (*Test-Driven Development*). 18
- UCI** Universidad de las Ciencias Informáticas. 18, 19, 25, b, 59, 60
- UML** Lenguaje de Modelado Unificado (*Unified Model Language*). 20, 21
- UP** Proceso Unificado (*Unified Process*). 18
- XML** Lenguaje Extensible de Marcas (*eXtensible Markup Language*). 12, 13

REFERENCIAS BIBLIOGRÁFICAS

- [Ambler(2006)] Scott W. Ambler. The agile unified process (aup) home page, 2006. URL <http://www.ambysoft.com/unifiedprocess/agileUP.html>.
- [ARM(2000)] ARM. Arm cortex-r series processors, 2000. URL <http://infocenter.arm.com/help/topic/com.arm.doc.set.cortexr/index.html>.
- [Beal(2012a)] Stephan Beal. s11.net: object serialization/persistence in c++, 2012a. URL <http://s11n.net/>.
- [Beal(2012b)] Stephan Beal. s11.net: libs11n: version 1.2, 2012b. URL <http://s11n.net/s11n/1.2/>.
- [Beal(2012c)] Stephan Beal. s11n.net: Serializers: s11n's link to the world of i/o:, 2012c. URL <http://s11n.net/serializers/>.
- [Beal(2013a)] Stephan Beal. s11n: easy object serialization in c++ | sourceforge.net, 2013a. URL <http://sourceforge.net/projects/s11n/>.
- [Beal(2013b)] Stephan Beal. s11n: easy object serialization in c++ / wiki / home, 2013b. URL <http://sourceforge.net/p/s11n/wiki/Home/>.
- [Boost(2004)] Boost. Serialization - boost, 2004. URL <http://www.boost.org/doc/libs/release/libs/serialization/>.
- [Boost(2014)] Boost. Boost c++ libraries. Sitio Web, 2014. URL <http://www.boost.org/>.
- [Codd(1970)] Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), Junio 1970.

- [Ding(2011)] Xiaoming Gu; Chen Ding. On the theory and potential of lru-mru collaborative cache management. *Communications of the ACM*, page 11, 2011.
- [Drepper(2007)] Ulrich Drepper. What every programmer should know about memory. *Communications of the ACM*, page 114, November 2007.
- [Eclipse(2014)] Eclipse. Eclipse desktop & web ide. Sitio Web, 2014. URL <https://www.eclipse.org/ide/>.
- [Edeki(2013)] Charles Edeki. Agile unified process. *International Journal of Computer Science and Mobile Applications 2013*, 1(3), 2013. ISSN 2321-8363.
- [Emer(2006)] Amer Jaleel; Kevin B. Theobald; Simon C. Steely Jr.; Joel Emer. High performance cache replacement using re-reference interval prediction (rrip). *Communications of the ACM*, 2006.
- [Española(2015)] Real Academia Española. Diccionario de la lengua española. 2015, 2015. URL <http://lema.rae.es>.
- [Flores(2009)] J. L. C. Ervin Flores. Metodologías Ágiles proceso unificado Ágil (aup). Página web: Sitio Oficial, Julio 2009. URL http://ingenieriadesoftware.mex.tl/63758_AUP.html.
- [Furuhashi(2013)] Sadayuki Furuhashi. Messagepack: It's like json. but fast and small., 2013. URL <http://msgpack.org/>.
- [Gamma(2006)] E. Gamma. *Patrones de diseño: elementos de software orientado a objetos re-utilizables.*, volume 1. P. Educación, 2006. URL <http://dspace.ucbscz.edu.bo/dspace/handle/123456789/565>.
- [GNU(2009)] GNU. Commoncpp - free software directory, 2009. URL <https://directory.fsf.org/wiki/Commoncpp>.
- [Google(2014a)] Google. Developer guide - protocol bbuffer - google developers, Septiembre 2014a. URL <https://developers.google.com/protocol-buffers/docs/overview>.
- [Google(2014b)] Google. Techniques - protocol bbuffer - google developers, 2014b. URL <https://developers.google.com/protocol-buffers/docs/techniques>.
- [Google(2014c)] Google. *Google C++ Testing Framework*. Google, 2014c. Fichero README.
- [Group(2001)] Object Management Group. Omg unified modmodel language specification, 2001. URL <http://www.omg.org>.

- [Guerrero and Suárez(2013)] C. A. Guerrero and J. M. Suárez. *Patrones de Diseño GOF (The Gang of Four) en el contexto de PProceso de Desarrollo de Aplicaciones Orientadas a la Web*. Información tecnológica, 2013.
- [Hamilton(2006)] Russ Miles Kim Hamilton. *Learning UML 2.0*. "Reilly Media, Inc.", 2006.
- [Hernández(2013)] Leovigilda Hernández. Modelo de implementación. Junio 2013, Junio 2013. URL <http://ithleovi.blogspot.com/2013/06/unidad-5-modelo-deimplementacion-el.html>.
- [Holzner(2006)] Steve Holzner. *Design Patterns for Dummies*. Wiley Publishing, Inc., 2006.
- [IBM(2011)] IBM. Information technology – database languages – sql – part 1: Framework (sql/framework), 2011. URL http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=53681. ISO/IEC 9075-1:2011.
- [Intel(2010)] Intel. Intel-cacheoverview. *Communications of the ACM*, page 10, 2010.
- [Intel(2014a)] Intel. Intel parallel studio xe 2015 | intel developer zone. Sitio Web, 2014a. URL <https://software.intel.com/en-us/intel-parallel-studio-xe>.
- [Intel(2014b)] Intel. Intel vtune amplifier 2015 | intel developer zone. Sitio Web, 2014b. URL <https://software.intel.com/en-us/intel-vtune-amplifier-xe>.
- [ISO(2014)] ISO. Iso/iec jtc1/sc22/wg21 - the c++ standards committee - isocpp. Sitio Web, 2014. URL <http://www.open-std.org/jtc1/sc22/wg21/>.
- [itiut(2014)] itiut. Messagepack specification, 2014. URL <https://github.com/msgpack/msgpack/blob/master/spec.md>.
- [Jacobson and Booch(2000)] I. Jacobson and G. Booch. *El proceso unificado de desarrollo de software.*, volume 7. Addison-Wesley Reading, 2000.
- [jwl(2010)] jwl. Pocerternitymanual - quimeraengine - quimera engine - google project hosting, 2010. URL <https://code.google.com/p/quimeraengine/wiki/POCEternityManual>.
- [KDevelop(2014)] KDevelop. Welcome to kdevelop.org. Sitio Web, 2014. URL <http://www.kdevelop.org>.
- [Larman(2003)] Craig Larman. *UML Y Patrones. Introducción al análisis y diseño orientado a objetos*. Prentice Hall, 2 edition, 2003.
- [Matani(2010)] Ketan Shah; Anirban Mitra; Dhruv Matani. An o(1) algorithm for implementing the lfu cache eviction scheme. *Communications of the ACM*, page 8, Agosto 2010.

- [Microsoft(2013)] Microsoft. Serialization in mfc, 2013. URL <https://msdn.microsoft.com/en-us/library/6bz744w8.aspx>.
- [Modha(2003)] Nimrod Megiddo; Dharmendra S. Modha. Arc: A self-tuning, low overhead replacement cache. *USENIX*, 2003.
- [Paradigm(2011)] Visual Paradigm. Software design tools for agile teams, with uml, bpmn and more, 2011. URL www.visual-paradigm.com.
- [Philbin(2001)] Yuanyuan Zhou; James F. Philbin. The multi-queue replacement algorithm for second level buffer caches. *USENIX*, page 15, 2001.
- [Postgres(2014)] Postgres. Postgresql: The world's most advanced open source database. Sitio Web, 2014. URL <http://www.postgresql.org/>.
- [Pressman(2010)] Ph. D. Roger S. Pressman. *Software Engineering A Practitioner's Approach Seventh Edition*. Mc Graw Hill Higher Education, 2010. ISBN 9780073375977.
- [Pressman(2005)] R. S. Pressman. *Ingeniería del Software, Un Enfoque Práctico*. España McGraw-Hil, 2005. ISBN 8448132149.
- [Reynoso(2004)] N. K. Carlos Reynoso. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft*. Universidad de Buenos Aires, 2004. URL <http://www.willydev.net/descargas/prev/Estiloypatron.pdf>.
- [Santi(2013)] Nicola Santi. Eternity / wiki / home, 2013. URL <http://sourceforge.net/p/eternity-it/wiki/Home/>.
- [Shasha(1994)] Theodore Johnson; Dennis Shasha. 2q: A low overhead high performance buffer management replacement algorithm. *VLDB*, 1994.
- [Solihin(2005)] Mazen Kharbutli; Yan Solihin. Counter-based cache replacement algorithms. *Communications of the ACM*, page 9, 2005.
- [Sommerville(2005)] lam Sommerville. *Requerimientos. En Ingeniería del Software.*, volume 7. Pearson Education S.A, 2005.
- [Stroustrup(2013)] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Professional, fourth edition edition, 2013. ISBN 0321563840, 9780321563842.
- [Stroustrup(2014)] Bjarne Stroustrup. C++ applications. Sitio Web, 2014. URL <http://www.stroustrup.com/applications.html>.

- [Suaza(2013)] Katerina Villamizar Suaza. Definición de equivalencias entre historias de usuario y especificaciones en un-lencep para el desarrollo ágil de software. *Communications of the ACM*, 2013. Facultad de Minas - Departamento de Ciencias de la Computación y de la Decisión. Universidad de Colombia.
- [Tamayo and Pulupa(2013)] F. R. Ortiz Tamayo and D. E. Alcocer Pulupa. *Diseño e implementación de un prototipo de un sistema computarizado distribuido orientado al control de la utilización de los servicios en un campus universitario a nivel local con capacidad para 2000 a 5000 estudiantes*. EPN, 2013.
- [Tan(1996)] Shaul Dar; Michael J. Franklin; Bjorn T. Jónsson; Divesh Srivastava; Michael Tan. Semantic data caching and replacement. *Communications of the ACM*, page 12, 1996.
- [Valgrind(2014)] Valgrind. Valgrind. Sitio Web, 2014. URL <http://valgrind.org/docs/manual/cl-manual.html>.
- [Voruganti(2004)] Kaladhar Voruganti. An adaptive data-shipping architecture for client caching data management systems. *Distrib. Parallel Databases*, 15, 2004.
- [Waters(2008)] John K Waters. Agile lands role in games and business software. *The Register*, 2008.