Universidad de las Ciencias Informáticas

Facultad 15

Título: Diseño del Motor de Procesamiento de Sentencias para el Sistema Chronos.

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

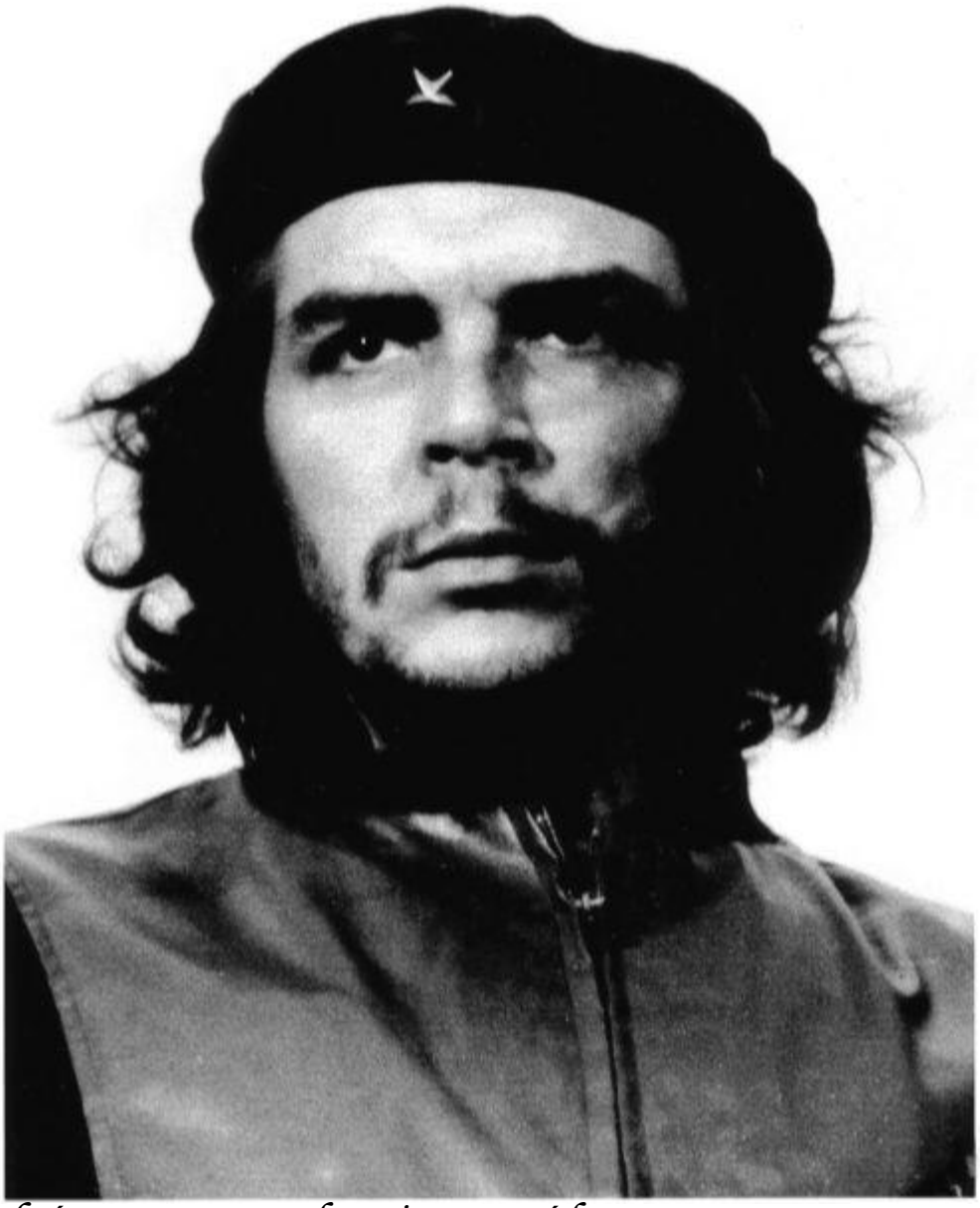
Autores: Tatiana Santana García

Diosnel Quiñones Rosales

Tutores: Ing. Yadira Lizama Mue

Ing. Yadisnel Gálvez Velázquez

Ciudad de la Habana, Junio 2010



"Si fuéramos capaces de unirnos, qué hermoso y que cercano sería el futuro"

Ernesto Che Guevara

Declaración de autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 15 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los _____ días del mes de _____ del año _____ .

Autores:	Tutores:
Tatiana Santana García	Ing. Yadira Lizama Mue
_____	_____
Firma del Autor	Firma del Tutor.
Diosnel Quiñones Rosales	Ing. Yadisnel Gálvez Velázquez
_____	_____
Firma del Autor	Firma del Tutor

Agradecimientos

A mis padres y mi familia, por siempre apoyarme en todo lo que he necesitado.

A Tatiana, por ayudarme en todo y compartir tantos momentos juntos.

A la UCI, por darme los conocimientos y valores necesarios para graduarme.

A los tutores, por hacer un gran trabajo con nosotros y soportarnos todo este tiempo.

Diosnel Quiñones Rosales

A mis padres por su inmenso amor y confianza en mí.

A mami por su ayuda, preocupación y apoyo a lo largo de la carrera. Gracias por guiarme el camino para salir adelante.

A papi por sus buenos consejos, confianza, por su apoyo incondicional que siempre me ayudaron a salir adelante y además por cocinarme la comida que me gusta.

A mi hermana Dalia a quien quiero mucho y por cuidar a mi mamita y a mi papito en todo este tiempo que apenas he podido estar con ellos.

A Diosnel por ser mi compañero de tesis, por su amistad, por su cariño, por su ayuda, por su comprensión, por estar siempre que lo he necesitado en todos estos años de Universidad.

A la Revolución Cubana y a Fidel por darme la oportunidad de estudiar en la Universidad del Futuro.

Quiero agradecer a los tutores Yadira y Yadisnel por su consagración y apoyo para que este trabajo resultara lo mejor posible.

A todos ustedes y los que de una forma u otra me apoyaron de todo corazón: "Mil Gracias".

Tatiana Santana García

Dedicatoria

De Diosnel

A mis padres, mi familia, mis amigos y a todos los que confiaron en mi.

A la revolución por hacer posible este sueño.

De Tatiana

A mi papito, a mi mamita, a mi hermanita Dalia por siempre apoyarme y confiar en mí.

A Diosnel por ayudarme siempre y por ser mi apoyo todos estos años.

Resumen

Los procesos de negocio involucran actualmente grandes volúmenes de información y la complejidad de sus relaciones se incrementa cada día, por lo que las empresas e instituciones científicas buscan alternativas eficientes y poco costosas que permitan el almacenamiento y tratamiento de la información. La replicación de los datos es una de las soluciones más utilizadas en la actualidad para este tipo de problemas. En la Universidad de las Ciencias Informáticas se desarrolla el Sistema Chronos, que pretende ser una herramienta de réplica asíncrona para entornos multimaestro. Ha sido concebida a partir de necesidades particulares de un escenario de réplica, asumiendo las principales potencialidades de las soluciones similares en la actualidad para Postgresql e implementando nuevas mejoras que la distinguen del resto. Surge la necesidad de las especificaciones del diseño de un componente que capture y replique las sentencias SQL originadas en un nodo del *cluster* de bases de datos al resto de los nodos de manera asíncrona, que recopile la información necesaria para realizar los reportes de mensajes de estados del *cluster* de bases de datos para las actividades de monitorización, que garantice la recuperación de los nodos en situaciones de *failover* y *failback* permitiendo la mayor convergencia de datos posible, que permita la replicación de datos instantánea y programada, una resolución de conflictos personalizada a partir de las exigencias de un negocio en particular, que cumpla con los tipos de sincronización del resto de las herramientas similares y que administre las conexiones a bases de datos de manera eficiente.

Palabras Claves: asíncrona, bases de datos, Chronos, multimaestro, Postgresql, réplica.

Índice general

Introducción

El establecimiento del conocimiento y la información, unido al incremento de los avances en las Tecnologías de la Informática y las Comunicaciones (TIC) y su rápida expansión a través de la Internet propician las condiciones necesarias para lograr obtener un desarrollo acelerado de las investigaciones. Los procesos de negocio involucran actualmente grandes volúmenes de información y la complejidad de sus relaciones se incrementa cada día, por lo que las empresas e instituciones científicas buscan alternativas eficientes y poco costosas que permitan el almacenamiento de la información, es por esto que surge la necesidad de la replicación de los datos. La réplica de bases de datos es una de las soluciones más generalizadas para alcanzar los retos que imponen la alta disponibilidad, tolerancia a fallos y escalabilidad en los sistemas de software actuales. La amplia gama de soluciones existentes incluyen sistemas que involucran replicación síncrona y asíncrona, desarrolladas para entornos de réplica multimaestro y maestro-esclavo. La replicación síncrona es muy utilizada en *cluster* de bases de datos destinados a garantizar servicios de alta disponibilidad, muy tolerante a fallos si es configurada en un entorno de réplica multimaestro. En entornos maestro-esclavo es aplicada donde el volumen de consultas de lectura posibles es superior a las transacciones de escritura o cuando es necesario centralizar los permisos de cambios en un servidor únicamente.

La replicación asíncrona es utilizada en sistemas donde la actualización de los cambios es un proceso independiente a la ejecución de la transacción en sí, lo que no significa que la réplica no se realice de manera instantánea. Si bien esto limita la utilidad para algunas configuraciones, la realidad es que en ocasiones se hace imprescindible su uso como la mejor solución a determinados escenarios. Las herramientas que permiten réplica de datos poseen diversas aplicaciones debido a su utilización para configurar sistemas que requieren alta disponibilidad de los datos, mayor capacidad de respuesta, balance de carga en el acceso y sistemas de copias de respaldo de datos, por solo mencionar los más importantes.

La réplica de datos supone un costo elevado en el tráfico de paquetes por la red y en la cantidad de operaciones que se ejecutan por cada transacción que realiza el usuario. Es necesario desarrollar pruebas que permitan seleccionar la mejor configuración de un entorno de réplica. Un estudio de las principales

herramientas de réplica para Postgresql existentes en la actualidad demostró la existencia de inconsistencias en situaciones de *failover* y *failback* en nodos del *cluster* de bases de datos, poca utilización de interfaces gráficas de usuario para la administración y configuración, la configuración total por *Shell*, la exigencia de un conocimiento a profundidad de la herramienta por el administrador de bases de datos, mínima validación de las configuraciones por lo que no se reduce el margen de error y la gran mayoría de las herramientas no permiten la configuración de réplica programada. En el caso de los sistemas asíncronos multimaestro presentan una resolución de conflictos muy básica.

En el país existe una política de migración de los sistemas de software hacia software libre y una de las opciones para los Sistemas de Gestión de bases de datos es Postgresql, la Base de Datos más avanzada de código abierto. En la Universidad de las Ciencias Informáticas (UCI) existe el Centro de Tecnologías de Almacenamiento y Análisis de Datos (CENTALAD) [1] que rige la política de migración en la Universidad y desarrolla un conjunto amplio de soluciones para la comunidad de Postgresql. La necesidad de sistemas de réplica unido al incremento de la utilización de Sistemas de Gestión de bases de datos de software libre en el país ha permitido que se incrementen los conocimientos sobre los sistemas de réplica para Postgresql, para resolver los problemas que se plantean en la actualidad, por lo que es de gran importancia crear una herramienta de réplica de datos para Postgresql.

En la Universidad de las Ciencias Informáticas se desarrolla el Sistema Chronos, que pretende ser una herramienta de réplica asíncrona para entornos multimaestro. Ha sido concebida a partir de necesidades particulares de un escenario de réplica, asumiendo las principales potencialidades de las soluciones similares en la actualidad para Postgresql e implementando nuevas mejoras que la distinguen del resto. Surge la necesidad de las especificaciones del diseño de un componente que capture y replique las sentencias SQL originadas en un nodo del *cluster* de bases de datos al resto de los nodos de manera asíncrona, que recopile la información necesaria para realizar los reportes de mensajes de estados del *cluster* de bases de datos para las actividades de monitorización, que garantice la recuperación de los nodos en situaciones de *failover* y *failback* permitiendo la mayor convergencia de datos posible, que permita la replicación de datos instantánea y programada , que permita una resolución de conflictos personalizada a partir de las exigencias de un negocio en particular, que cumpla con los tipos de

sincronización del resto de las herramientas similares y que administre las conexiones a bases de datos de manera eficiente .

A partir del análisis de esta **situación problemática** surge la presente investigación que se plantea el **problema** de: Inexistencia de un componente que permita la réplica asíncrona de sentencias y la recopilación de información para actividades de monitorización en un *cluster* de bases de datos Postgresql. El **objeto de estudio** en el que se enmarca la investigación lo constituyen los procesos de réplica de bases de datos objeto-relacionales. El **campo de acción** abarcado son los procesos de réplica asíncrona multimaestro de bases de datos Postgresql.

El **objetivo general de esta investigación es** elaborar el diseño del Motor de Procesamiento de Sentencias del Sistema Chronos.

Los **objetivos específicos** son**:**

1- Desarrollar el marco teórico de la investigación del diseño del Motor de Procesamiento de Sentencias del Sistema Chronos.

2- Identificar los procesos relacionados con la réplica en el Sistema Chronos.

3- Modelar el Diseño del Motor de Procesamiento de Sentencias del Sistema Chronos.

4- Validar utilizando técnicas adecuadas para comprobar la calidad del diseño realizado.

Para guiar la investigación propuesta se plantea la siguiente **Idea a defender:**

- Con la modelación del diseño del Motor de Procesamiento de Sentencias del Sistema Chronos se alcanzará transformar los procesos de réplica de este sistema a un lenguaje entendible por los desarrolladores para su posterior implementación.

Las **tareas** que se llevarán a cabo para dar cumplimiento a los objetivos trazados son las siguientes:

- Caracterizar los procesos de réplica en Sistemas de Gestión de Bases de Datos Objeto-Relacionales.
- Analizar los requerimientos de réplica de bases de datos en sistemas actuales.
- Caracterizar la arquitectura propuesta para el Sistema Chronos.
- Caracterizar las tecnologías y herramientas existentes para réplica de bases de datos Postgresql.
- Análisis y selección de los patrones de diseño a utilizar.
- Análisis y selección de las métricas a utilizar.
- Elaborar la propuesta de solución del Diseño del Motor de Procesamiento de Sentencias del Sistema Chronos.
- Análisis y selección de las herramientas, metodologías y tecnologías a utilizar para la elaboración de la propuesta de solución.
- Especificar los requerimientos de la réplica de bases de datos en el Sistema Chronos.
- Diseño del Motor de Procesamiento de Sentencias para el Sistema Chronos.
- Validar el diseño del Motor de Procesamiento de Sentencias del Sistema Chronos.

El presente trabajo consta de 4 capítulos, Introducción, Conclusiones, Recomendaciones, Bibliografía, Anexos y un Glosario de Términos.

En el **Capítulo 1 Fundamentación Teórica**: se plantea la fundamentación teórica del tema, incluye un estudio del estado del arte abordando temas como las últimas tendencias, técnicas, metodologías y herramientas que se usan en la actualidad para la réplica en bases de datos Postgresql. Todo esto a nivel internacional y nacional, realizando énfasis en la Universidad de las Ciencias informáticas como el principal desarrollador de software de Cuba en la actualidad.

En el **Capítulo 2 Características del sistema**: se describe el sistema que se propone para dar solución a la problemática actual que da origen al mismo, así como la descripción de los procesos que serán

automatizados. Se recogen las características básicas del sistema expresadas en funcionalidades y en requisitos no funcionales como soporte, rendimiento, confiabilidad, entre otras.

En el **Capítulo 3 Diseño del sistema**: se realiza el diseño de la solución propuesta. Esto incluye los diagramas de interacción que representan el flujo principal y alternativo de eventos en cada caso de uso del sistema. Se describen y se realizan los diagramas de clases, así como la descripción de las tablas de la Base de Datos y el diseño de la misma.

En el **Capítulo 4 Validación del Diseño**: se aplican una serie de métricas de diseño orientado a objetos y se analizan los resultados para determinar la calidad del diseño realizado para el Motor de Procesamiento de Sentencias del sistema Chronos.

Capítulo 1. Fundamentación Teórica

Introducción

En este capítulo se aborda el estado del arte de la réplica de datos para Postgresql en el ámbito nacional e internacional, así como en la Universidad de las Ciencias Informáticas. Se incluyen las tendencias actuales, las técnicas más empleadas así como las tecnologías y metodologías utilizadas en esta investigación. Se describen las características de las herramientas de réplica más utilizadas para Postgresql, ventajas, desventajas y capacidades de rendimiento con el objetivo de establecer los fundamentos teóricos en los que se basa la propuesta de solución.

1.1 Réplica de datos

Para comprender el objeto de estudio del trabajo se fundamentan algunos conceptos importantes sobre el contenido de esta investigación.

¿Qué es un Sistema de Gestión de Bases de Datos (SGBD)?

"*Un SGBD es la herramienta que permite interactuar los datos con los usuarios de los datos, de forma que se garanticen todas las propiedades definidas en una base de datos*" [2]. Estas propiedades son la integridad, confidencialidad, seguridad y disponibilidad de los datos. Actualmente estos sistemas presentan problemas con el procesamiento de gran volumen de información, por lo que surge la necesidad de la replicación de datos como solución, para lograr un mejor rendimiento del SGBD.

¿Qué es un *cluster*?

Un *cluster* consiste en un tipo de sistema de procesamiento paralelo o distribuido, compuesto por un conjunto de computadoras que trabajan cooperativamente como un único e integrado recurso de cómputo [3].

Los conceptos de SGBD y de *cluster* están estrechamente relacionados con la réplica de datos, que según [4] se define como: "*En un sistema de gestión de bases de datos distribuidas, es el proceso de copiar la base de datos (o partes de la misma) a los otros lugares de la red. La replicación permite a los sistemas de bases de datos distribuidas mantener la sincronización".*

Este proceso asegura que los datos estén siempre disponibles en el lugar necesario para ser utilizados en el momento indicado.

1.2 Tipos de réplica

Los tipos de replicación están relacionados con el momento en que se realiza la actualización de los datos cuando se efectúa una transacción de escritura en la Base de Datos.

1.2.1 Replicación Síncrona

En el caso de la replicación síncrona, si se actualiza una réplica dada, todas las demás réplicas del mismo fragmento de datos también se actualizan dentro de la misma transacción; lo que implica que (desde un punto de vista lógico) solo existe una versión de los datos. La mayoría de los productos implementan la replicación síncrona por medio de disparadores (posiblemente ocultos y manejados por el sistema). Sin embargo la replicación síncrona tiene la desventaja de que impone una sobrecarga sobre las transacciones que actualizan cualquier réplica [5].

1.2.2 Replicación Asíncrona

En el caso de la replicación asíncrona, las actualizaciones a una réplica son propagadas hacia los demás en algún momento posterior, no dentro de la misma transacción, por lo tanto la replicación asíncrona presenta un retardo de tiempo o latencia, durante el cuál es posible que las réplicas no sean idénticas [5].

1.3 Entornos de Réplica

La réplica de datos puede aplicarse de dos maneras diferentes de acuerdo a las necesidades propias de las personas, sistemas o instituciones que la utilicen.

1.3.1 Multimaestro

El entorno de replicación multimaestro es conocido como “par a par o la réplica de camino de n” que permite múltiples nodos actuando como pares iguales. En este entorno cada nodo es maestro, y se comunica con otros nodos maestros. Cada nodo permite operaciones de lectura y escritura, cuando se modifica cualquiera de los nodos el resto es actualizado, por lo que se elimina la sobrecarga de un único nodo. La replicación multimaestro es una alternativa eficaz para aquellos sistemas que realizan gran cantidad de operaciones de lectura/escritura sobre sus datos [6].

1.3.2 Maestro-Esclavo.

Entorno de replicación maestro-esclavo es conocida también como “de solo lectura “, permite a un solo maestro recibir consultas de lectura/escritura, mientras los esclavos solo pueden aceptar consultas de lectura. Debido a que las operaciones de escritura pueden ser efectuadas en un único nodo, para sistemas que realicen modificaciones constantemente, no se logran buenos índices de rendimiento [7].

1.4 Técnicas de replicación asíncrona

Las técnicas de replicación asíncrona están relacionadas con el intervalo de tiempo que existe entre un cambio realizado en la base de datos y el momento en que este se replica.

1. La replicación instantánea ocurre cuando el intervalo de tiempo entre el “commit” de una transacción SQL y el momento en que se realiza la réplica hacia el resto de las fuentes de datos es muy pequeño. Es utilizada cuando se desea obtener la sincronización del sistema en el menor tiempo posible [8] [9].
2. La replicación programada se utiliza para sistemas donde la sincronización de los datos no tiene que ser inmediata y puede realizarse en determinados momentos en los que el *cluster* de bases de datos no esté recibiendo tanto volumen de carga. Esta puede ser configurada con una periodicidad diaria, semanal, mensual o cada N horas. Siendo una de las características del sistema que permite su flexibilidad para escenarios donde existan estaciones de trabajo locales que no necesiten de constante actividad con los servidores centrales [8] [9].

1.5 Sistemas de Réplica para Postgresql en el mundo.

En el ámbito internacional existen varias herramientas desarrolladas por la comunidad de Postgresql. Entre estas se destacan Pg*Cluster*, Slony, Pgpool y Bucardo.

1.5.1 Herramientas de réplica para entornos de replicación síncronos.

1.5.1.1 Pgpool-II

Pgpool-II es un (*middleware*) que trabaja entre los servidores de Postgresql y un cliente de bases de datos Postgresql. Pgpool-II guarda conexiones en los servidores Postgresql, para volver a utilizarlas cuando se realice una nueva conexión con las mismas propiedades (usuario, base de datos, versión del protocolo). Esto reduce gastos en términos de conexión y mejora el rendimiento general del sistema. Puede administrar varios servidores Postgresql. La utilización de la función de replicación permite crear una copia de seguridad en tiempo real sobre dos o más discos físicos, de modo que el servicio pueda seguir sin detener los servidores en caso de un fallo de disco [10].

1.5.1.2 PGCluster

PGCluster es una herramienta de código abierto, basada en modificaciones al núcleo de Postgresql que permite la replicación síncrona multimaestro de bases de datos. Este tipo de herramientas de réplica es muy utilizado para configurar sistemas de alta disponibilidad de datos [11]. Los servidores involucrados pueden ser de tres tipos diferentes respecto a las actividades de réplica: Servidor de Balance de Carga, Servidor de Réplica y Servidor de bases de datos. Respecto a las actividades de monitorización y administración, existe un Servidor Admin y el resto de los servidores son Probe.

1.5.2 Herramientas de réplica asíncrona

1.5.2.1 Slony-I

Slony-I es un sistema de replicación asíncrona maestro-esclavo, que incluye el tipo de capacidades necesarias para replicar bases de datos grandes a un número razonablemente limitado de los sistemas

esclavos del orden de una docena de servidores. Si el número de servidores crece más allá de eso, el costo prohibitivo de las comunicaciones aumenta, y puede ocasionar problemas en el *cluster* para la réplica de datos. Slony-I es un sistema diseñado para centros de datos y sitios de seguridad informática donde se necesita de una alta disponibilidad del servicio en varios nodos [12].

1.5.2.2 Bucardo

Bucardo es un sistema de replicación asíncrona para Postgresql, soporta la configuración multimaestro y maestro-esclavo. Fue desarrollado en Backcountry.com por Jon Jensen y Greg Sabino Mullane. Está escrito en Perl, y utiliza un amplio uso de disparadores, PL / pgSQL y PL / perlu. Bucardo, requiere de una Base de Datos para instalar el esquema de Bucardo principal. Esta Base de Datos Postgresql debe ser la versión 8.1 o superior, y debe tener los dos idiomas de PL / pgSQL y PL / perlu disponibles. Bucardo no requiere hardware especial y permite múltiples servidores maestros. Además los servidores esclavos aceptan consultas de solo lectura [13].

1.6 Sistemas de Réplica para Postgresql desarrolladas en Cuba y la UCI

Dentro de las soluciones propuestas para la réplica de datos en nuestro país y en la UCI se encuentran:

- Reko solución para la réplica de datos del Proyecto Sistema de Gestión Penitenciaria (SIGEP).
- Réplica bidireccional basada en control de cambios del Proyecto Identidad.

1.6.1 Reko: Solución para la réplica de datos del Proyecto Sistema de Gestión Penitenciaria (SIGEP).

Reko soporta la replicación de transacciones a través de Hibernate y la replicación de acciones a través de disparadores. Se integra actualmente con los gestores de bases de datos Oracle y Postgresql. Provee la facilidad de seleccionar el subconjunto de tablas y datos a ser replicados de la base de datos mediante la definición de filtros aplicables a las tablas y la selección de usuarios propios del gestor. La transferencia de datos de replicación puede realizarse mediante soporte para replicación sobre TCP/IP, FTP, HTTP o por ficheros de forma manual. La administración y configuración de la réplica se realiza a través de una

interfaz Web, por lo que es administrable vía remota usando solamente un navegador. Mantiene los datos de réplica en un estado estable en caso de desconexión. Al restablecerse la conexión, automáticamente sincroniza los datos entre las bases de datos. No realiza la técnica de réplica programada, posee una resolución de conflictos muy básica y no administra de forma inteligente las conexiones a la base de datos [14].

1.6.2 Réplica bidireccional basada en control de cambios.

Esta solución está propuesta por el proyecto Identidad de la Universidad de las Ciencias Informáticas. Se caracteriza fundamentalmente por ser bidireccional, basado en control de cambios y no en colas, lo que evita las transmisiones redundantes. Soporta ambientes heterogéneos, no importa el sistema de gestión de base de datos. Soporta particionado horizontal de los datos y garantiza la integridad de los mismos, pues provee mecanismos de detección y resolución de conflictos. Está implementado sobre la plataforma .NET y posee una topología flexible y escalable. No soporta el entorno de replicación maestro-esclavo, no realiza la técnica de réplica programada, posee una resolución de conflictos muy básica y no administra de forma inteligente las conexiones a la base de datos [1].

1.7 Tecnologías y herramientas de software para el desarrollo.

1.7.1 Postgresql

Postgresql es un poderoso sistema de gestión de bases de datos objeto-relacional de código abierto. Postgresql es un sistema objeto-relacional, ya que incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional [15]. A pesar de esto, Postgresql no es un sistema de gestión de bases de datos puramente orientado a objetos. Es compatible con todos los Sistemas Operativos, Incluyendo Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), y Windows. Es totalmente compatible con ACID, tiene soporte completo para claves foráneas, uniones, vistas, ejecuta procedimientos almacenados en más de una docena de idiomas de programación.

1.7.2 Visual Paradigm.

Es una herramienta de modelado profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Es multiplataforma, muy potente y fácil de utilizar. Puede generar código automático, bases de datos y generar reportes. Permite la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso. Permite la generación de código y despliegue de EJB´s y la generación de beans para el desarrollo y despliegue de aplicaciones, diagramas de flujo de datos. Posee un generador de informes para generación de documentación [16] [17].

1.7.3 Lenguaje Unificado de Modelado (UML).

El Lenguaje Unificado de Modelado (UML) es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra gran cantidad de software. UML permite una forma de modelar cosas conceptuales como lo son procesos de negocio y funciones de sistema, además de cosas concretas como lo son escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reusables. UML permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos [18] [19].

1.7.4 Metodología de desarrollo de software: *Rational Unified Process*.

RUP (Proceso Unificado de Desarrollo), llamada así por sus siglas en inglés *Rational Unified Process*, es un proceso de desarrollo de software. RUP es un proceso para el desarrollo de un producto que define claramente quién, cómo, cuándo y qué debe hacerse en el proyecto. Esta metodología constituye una guía rectora que dirige las actividades que se realizan por rol, les da un orden, especifica qué artefactos deberían ser desarrollados y puede además monitorear y medir los productos y actividades de un proyecto, teniendo en cuenta la calidad del producto final. Las características esenciales de RUP son que es dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental. RUP posee múltiples ventajas como son la valoración en cada fase que permite cambios de objetivos, permite buen funcionamiento en proyectos de creación así como el análisis y seguimiento detallado en cada una de las

fases. Entre las desventajas que posee se encuentra que la evaluación de riesgos es compleja y el cliente debe ser capaz de describir y entender a un gran nivel de detalle para poder acordar un alcance del proyecto con él [20].

1.8 Métricas para validar el Diseño

Las métricas para validar el diseño proporcionan al diseñador una mejor visión interna, ayudan a que el diseño evolucione a un nivel superior de calidad. Para la aplicación de estas métricas Berard Laranjeira define cinco características fundamentales estas son la localización, el encapsulamiento, ocultamiento de la información, la herencia y las técnicas de abstracción de objetos [21].

Familia de métricas propuestas por Chidamber y Kemener

Los conjuntos de métricas propuestos por Chidamber y Kemener han sido uno de los más referenciados [21], las que son conocidas normalmente como la serie de métricas CK. Chidamber y Kemener proponen seis métricas basadas en clases para sistemas orientados a objetos: métodos ponderados por clase, profundidad árbol de herencia, número de descendientes, acoplamiento entre clases, respuesta para una clase y carencia de cohesión en los métodos.

Métricas propuestas por Lorenz y Kidd

Lorenz y Kidd dividen las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos [21]. Las métricas orientadas a tamaños para una clase se centran en cálculos de atributos y de operaciones para una clase individual, y promedian los valores para el sistema en su totalidad. Las métricas basadas en herencia se centran en la forma en que se reutilizan las operaciones a lo largo y ancho de la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión y asuntos relacionados con el código, y las métricas orientadas a valores externos examinan el acoplamiento y la reutilización.

1.8.1 Tamaño de Clase (TC)

Lorenz y Kidd proponen que para medir el tamaño de clases se deben tener los siguientes aspectos en cuenta:

- ✓ Total de operaciones, ya sean las de la clase o las heredadas de las clases padres o interfaces que implementen.
- ✓ Cantidad de atributos, al igual que el anterior, tanto los de ella, como los de los padres.
- ✓ Promedio de operaciones y atributos para el sistema completo.

Si existen valores grandes de TC éstos mostrarán que una clase puede tener demasiada responsabilidad, lo cual reducirá la reutilización de la clase y complicará la implementación y la comprobación, por otra parte cuanto menor sea el valor medio para el tamaño, más probable es que las clases existentes dentro del sistema se puedan reutilizar ampliamente.

1.8.2 Relaciones entre Clases (RC).

Esta métrica está dada por la cantidad de relaciones de uso que existe entre las distintas clases que forman el diseño propuesto. Los aspectos de calidad que se miden son: Acoplamiento, Complejidad de mantenimiento, Reutilización y Cantidad de pruebas.

1.8.3 Tamaño operacional de clase (TOC)

Esta métrica está dada por el número de métodos asignados a una clase. Los aspectos de calidad que se miden son: Responsabilidad, Complejidad de mantenimiento y Reutilización.

De todas estas métricas estudiadas las escogidas para la validación del diseño fueron Tamaño de Clase (TC), Tamaño operacional de clase (TOC) y Relaciones entre Clases (RC) ya que son las que aportan mayor información a la validación del diseño propuesto y permiten realizar la validación de que los patrones Alta Cohesión y Bajo Acoplamiento fueron bien utilizados.

Conclusiones

En este capítulo se han introducido conceptos indispensables para la comprensión de los procesos de réplica de bases de datos. Se propone realizar una nueva solución donde se utilice como Sistema de Gestión de bases de datos Postgresql ya que es el gestor de software libre más potente en la actualidad. Se escoge como herramienta para el modelado el Visual Paradigm, ya que soporta muy bien la metodología RUP, que es la que se utilizará y el lenguaje de modelación visual UML. La metodología propuesta para el desarrollo de la aplicación es RUP, por su capacidad para guiar a los trabajadores a desarrollar un programa, por ser iterativo y tener como objetivo la entrega gradual de soluciones. Se analizan y seleccionan las métricas para validar el diseño.

Capítulo 2. Características del sistema.

Introducción

En el presente capítulo se hace la descripción de la propuesta de solución, se describen los procesos a automatizar así como las características del sistema. También se realiza un análisis comparativo de otras soluciones existentes con la propuesta así como la especificación de los requisitos de software en funcionales y no funcionales y se definen los casos de uso del sistema.

2.1 Descripción del objeto de estudio

Problema y situación problemática:

A partir del estudio realizado de las principales herramientas de réplica para Postgresql se detectaron algunas deficiencias como la existencia de inconsistencias en situaciones de *failover* y *failback* en nodos del *cluster* de bases de datos, poca utilización de interfaces gráficas de usuario para la administración y configuración: la configuración total por Shell, la exigencia de un conocimiento a profundidad de las herramientas por el administrador de bases de datos, mínima validación de las configuraciones por lo que no se reduce el margen de error y la gran mayoría no permite la configuración de réplica programada. En el caso de los sistemas asíncronos multimaestro presentan una resolución de conflictos muy básica, destacándose Bucardo como la única herramienta que proporciona este tratamiento de conflictos.

Es por esto que surge la necesidad de las especificaciones del diseño de un componente que capture y replique las sentencias SQL originadas en un nodo del *cluster* de bases de datos al resto de los nodos de manera asíncrona, que recopile la información necesaria para realizar los reportes de mensajes de estados del *cluster* de bases de datos para las actividades de monitorización, que garantice la recuperación de los nodos en situaciones de *failover* y *failback* permitiendo la mayor convergencia de datos posible, que permita la replicación de datos instantánea y programada , que permita una resolución de conflictos personalizada a partir de las exigencias de un negocio en particular, que cumpla con los tipos de sincronización del resto de las herramientas similares y que administre las conexiones a bases de datos de manera eficiente.

De ahí que surge el problema de la inexistencia de un componente que permita la réplica asíncrona de sentencias y la recopilación de información para actividades de monitorización en un *cluster* de bases de datos Postgresql.

Objeto de automatización.

Los procesos a automatizar son los siguientes:

Réplica instantánea de transacciones entre bases de datos Postgresql: Este proceso permite la replicación instantánea de transacciones entre bases de datos Postgresql. A este tipo de replicación se hace referencia en el epígrafe 1.4 de esta investigación.

Réplica programada de transacciones entre bases de datos Postgresql: Este proceso permite la replicación programada de transacciones entre bases de datos Postgresql. A este tipo de replicación se hace referencia en el epígrafe 1.4 de esta investigación.

Recopilación de información del estado del *cluster* de bases de datos: Este proceso se encarga del envío de mensajes sobre la información de la réplica en su nivel más bajo. Se encarga del registro de la actividad de réplica en cada nodo de bases de datos mediante ficheros de log. La información es más específica y detallada y consiste en: usuarios, réplicas realizadas, estados de consistencia de las bases de datos, estado de las sincronizaciones programadas y tipos de eventos que el sistema está procesando.

Administración de Conexiones a las bases de datos del *cluster*: Este proceso se encarga de administrar las conexiones a las bases de datos del *cluster*, permitiendo el uso eficiente de las mismas y aumentando el rendimiento general de los nodos.

Carga de información de eventos de réplica: Este proceso se encarga de cargar toda la información de los eventos programados e instantáneos configurados para ejercer las actividades de réplica entre bases de datos.

2.2 Propuesta de sistema.

Se propone el desarrollo de las especificaciones del diseño para un sistema que permita la replicación instantánea y programada, la resolución de conflictos personalizados, la utilización de los tipos de sincronizaciones, la administración de conexiones inteligentes y la recuperación ante fallos de los nodos del *cluster* de Base de Datos.

Componentes del Sistema

El sistema Chronos está integrado por varios componentes que permiten el funcionamiento de la réplica y la integración con el Sistema de Gestión de bases de datos (SGBD) Postgresql. En la figura 1 se representa un diagrama de componentes generales del mismo.

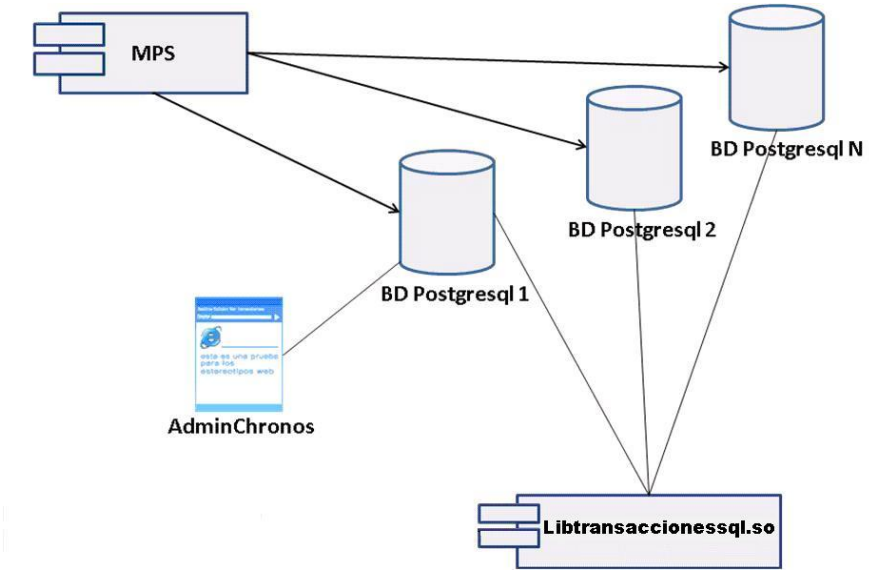


Figura 1. Componentes del Sistema Chronos

BD Postgresql desde 1 hasta N: Versiones liberadas de la base de datos Postgresql que constituyen los Sistemas de Gestión de bases de datos con los que interactúa el Sistema Chronos **Libtransaccionesql.so**: Librería de enlace dinámico que extiende la funcionalidad del gestor capturando los cambios que se producen en cualquiera de las tablas contenidas, los cuales deben ser replicados según el tipo de sincronización que se configure.

Motor de Procesamiento de Sentencias (MPS): El Motor de Procesamiento de Sentencias (MPS) deberá replicar los cambios ocurridos en los Sistemas de Gestión de Base de Datos (SGBD) Postgresql configurados por la herramienta de administración (AdminChronos) y utilizará la librería Libtransaccionesql.so para capturar los cambios que se producen en cualquiera de las tablas de la Base de Datos.

AdminChronos: Aplicación Web de administración, configuración y monitorización en línea del sistema. Posee una interfaz gráfica de usuario amigable y fácil de usar, lo que le confiere un valor agregado a la aplicación. A través de esta es posible realizar todas las configuraciones de sincronización para todos los nodos del *cluster* de manera sencilla y rápida. Permite monitorizar el estado de la réplica haciendo interactuar al usuario con este como parte del sistema.

2.3 Características del Sistema Chronos

El funcionamiento de estos componentes debe garantizar un conjunto de características en el sistema que lo diferencian del resto de las herramientas de su tipo:

2.3.1 Soporte para multiprocesamiento

El Motor de Procesamiento de Sentencias (MPS) debe constituir el núcleo de réplica de la aplicación y permitir soporte para multiprocesamiento a partir de técnicas avanzadas de programación concurrente. Esto permitirá lograr una mayor cantidad de Transacciones Por Segundo (TPS) aumentando el rendimiento total del sistema a la vez que aprovecha las características multinúcleo de los servidores. Además se deberá ejecutar como un proceso independiente capturando, planificando y replicando los cambios ocurridos en los SGBD Postgresql.

2.3.2 Técnicas de Replicación

Las técnicas de réplica asumidas por el servidor deben ser instantáneas o programadas, cumpliendo con las exigencias de este tipo de software en la actualidad.

2.3.3 Resolución de conflictos personalizados

El sistema Chronos debe ofrecer un conjunto de soluciones estándares frecuentes para los conflictos que pueden surgir durante el proceso de sincronización de los datos. Si las soluciones no satisfacen las necesidades del usuario, este podrá definir sus propios métodos de solución de conflictos que pueden ser muy específicos del negocio al que responde la Base de Datos, permitiendo la adaptabilidad del sistema a escenarios específicos de réplica.

2.3.4 Tipos de Sincronizaciones

Las sincronizaciones entre fuentes de datos definen cómo se realizará la réplica entre una fuente origen y su destino. Estarán clasificadas en tres tipos diferentes:

1. **Adicionar Diferencias:** Permite adicionar sólo las diferencias (tuplas o registros) de una fuente de datos origen hacia una fuente de datos destino.
2. **Intercambio:** Garantiza que los datos de una fuente origen se adicionen en la fuente destino y viceversa. Esta sincronización requiere un método de resolución de conflictos que permita tomar una decisión en caso de que ocurra una anomalía. Es muy utilizado en escenarios de réplica bidireccional.
3. **Copia Total:** Este método establece una copia total de una fuente de datos origen hacia una fuente destino. Los datos de la fuente destino se pierden y son reemplazados por la tabla origen. Suele ser bastante costoso para la réplica instantánea y es muy utilizado para recuperaciones de fuentes de datos que inician desde cero o adición de nuevas fuentes a sincronizaciones ya establecidas.

2.3.5 Administración de conexiones inteligente

Chronos manipulará de manera inteligente las conexiones a las bases de datos del *cluster*, permitiendo el uso eficiente de las mismas y aumentando el rendimiento general de los nodos.

2.3.6 Recuperación ante fallos

Ante la caída de un nodo el sistema debe ser capaz de detectarlo. Cuando se restablece el servicio, Chronos reiniciará una recuperación de las bases de datos inconsistentes, logrando la convergencia de los datos respecto al resto de los nodos.

2.4 Análisis comparativo de otras soluciones existentes con la propuesta

La replicación instantánea es implementada por todas las herramientas de réplica existentes en el mundo como Pgpool II, Bucardo, Reko, Réplica bidireccional basada en control de cambios, Slony-I y Pg*Cluster* y la propuesta de solución implementará la replicación programada que es totalmente nueva además de la instantánea. Destacar que Bucardo es la única herramienta de réplica que utiliza la resolución de conflictos personalizados pero de manera muy pequeña y los tipos de sincronizaciones, la resolución de conflictos personalizados que se prevee en la propuesta es bastante similar a la que utiliza Bucardo pero de manera más amplia y utilizará los tipos de sincronizaciones. Las herramientas de réplica existentes no permiten administración de conexiones inteligentes, excepto la herramienta de réplica Pgpool II y no permiten la recuperación ante fallos de manera automática, excepto la herramienta de réplica Pg*Cluster*, la propuesta de solución permitirá ambas características. Las herramientas de réplica Reko y Réplica bidireccional basada en control de cambios, no realizan la técnica de réplica programada, poseen una resolución de conflictos muy básica y no administran de forma inteligente las conexiones a la base de datos, la propuesta de solución reúne todas estas características.

2.5 Dependencias y Relaciones con otros software

Para el correcto funcionamiento del Motor de Procesamiento de Sentencias (MPS) es necesario utilizar las librerías que se describen a continuación:

Libconfuse (Lectura de parámetros de configuración)

Libconfuse es una librería para el análisis de los archivos de configuración escritos en C. Soporta secciones, listas, valores como cadenas, enteros, reales, booleanos entre otros, así como algunas otras características tales como uno o dos cadenas entre comillas, la expansión de variable de entorno,

funciones e incluye también declaraciones anidadas. Es utilizada por el Lector de Parámetros de configuración para parsear los datos configurados en el fichero chronos.conf [22].

Libpq (conexión a postgres desde C++)

Libpq es una librería para la conexión a Postgresql desde C++. Esta permite que los programas de usuario puedan comunicarse con la Base de Datos Postgresql. La Base de Datos Postgresql puede ser local o puede ser en otra máquina y acceder a través de TCP / IP. Se utiliza para la comunicación con las bases de datos en réplica desde el MPS [23].

2.6 Especificación de los requisitos de software

El motor de procesamiento de sentencias debe cumplir una serie de aspectos desde el punto de vista funcional. Además se regirá por otros aspectos desde el punto de vista no funcional como rendimiento, soporte y confiabilidad.

2.6.1 Requisitos funcionales

RF1. **Ejecutar réplica instantánea.**

RF1.1 Buscar configuración de eventos.

RF1.2 Buscar Base de Datos orígenes por eventos.

RF1.3 Buscar fuentes destinos por evento.

RF2. **Ejecutar réplica programada.**

RF2.1 Buscar configuración de eventos.

RF2.2 Buscar Base de Datos orígenes por eventos.

RF2.3 Buscar fuentes destinos por eventos.

RF3. **Replicar sentencia.**

RF4 **Cargar configuración de eventos.**

RF4.1 Validar Configuración de eventos.

RF5. **Emitir notificación.**

RF5.1 Registrar Log.

RF6. **Gestionar conexiones a la Base de datos.**

RF6.1 Crear conexiones a la Base de Datos.

RF6.2 Cerrar conexión a la Base de Datos.

RF6.3 Verificar estado de conexión a la Base de Datos.

RF6.4 Eliminar conexión de la Base de Datos.

RF6.5 Modificar parámetros de conexión a la Base de Datos.

RF6.6 Activar conexión a la Base de Datos.

RF7. **Recuperar ante fallos.**

RF8. **Cargar parámetros de configuración.**

RF9. **Reportar estado del MPS.**

RF10. **Restaurar fuente con copia total.**

2.6.2 Requisitos no funcionales.

RNF1 Rendimiento: El sistema deberá tener un nivel de respuesta aceptable, tanto para el acceso a bases de datos, como para el proceso de réplica de datos, no mayor de 1 segundo por transacción.

RNF2 Soporte: El sistema necesita la instalación de las siguientes librerías para su correcto funcionamiento: Libconfuse y Libpq.

RNF3 Portabilidad: Debe soportar como Sistema de Gestión de Bases de Datos (SGBD) Postgresql y será portable en sistemas operativos con plataforma Linux Debian/GNU.

RNF4 Confiabilidad: Debe ser confiable en un alto grado, dado su fin de replicar la información. Debe garantizar la convergencia de los datos, sin ocasionar pérdida de información durante la réplica, ni inconsistencias

2.7 Definición de los casos de uso.

2.7.1 Definición de los actores.

Tabla 1.Descripción de los actores

Actores	Justificación
mps	El mps es un actor ficticio que realiza la mayoría de las funcionalidades del sistema. Inicia la réplica instantánea, restaura fuente con copia total, carga los parámetros de configuración, gestiona conexiones a bases de datos y carga la configuración de eventos.
reloj	Inicia la réplica programada cuando ocurre en el tiempo el suceso configurado e inicia el reporte de estado del MPS.
red	Inicia el proceso de recuperación de bases de datos desconectadas, cuando se restablece la conexión.

2.7.2 Definición de los casos de uso

Tabla 2. Definición de los casos de uso

Caso de uso	Nombre
CU-1	Ejecutar réplica instantánea
CU-2	Ejecutar réplica programada
CU-3	Replicar sentencia
CU-4	Cargar configuración de eventos
CU-5	Emitir notificación
CU-6	Gestionar conexiones a la Base de datos.
CU-7	Recuperar ante fallos
CU-8	Cargar parámetros de configuración
CU-9	Reportar estado del MPS
CU-10	Restaurar fuente con copia total

2.7.3 Diagrama de casos de uso.

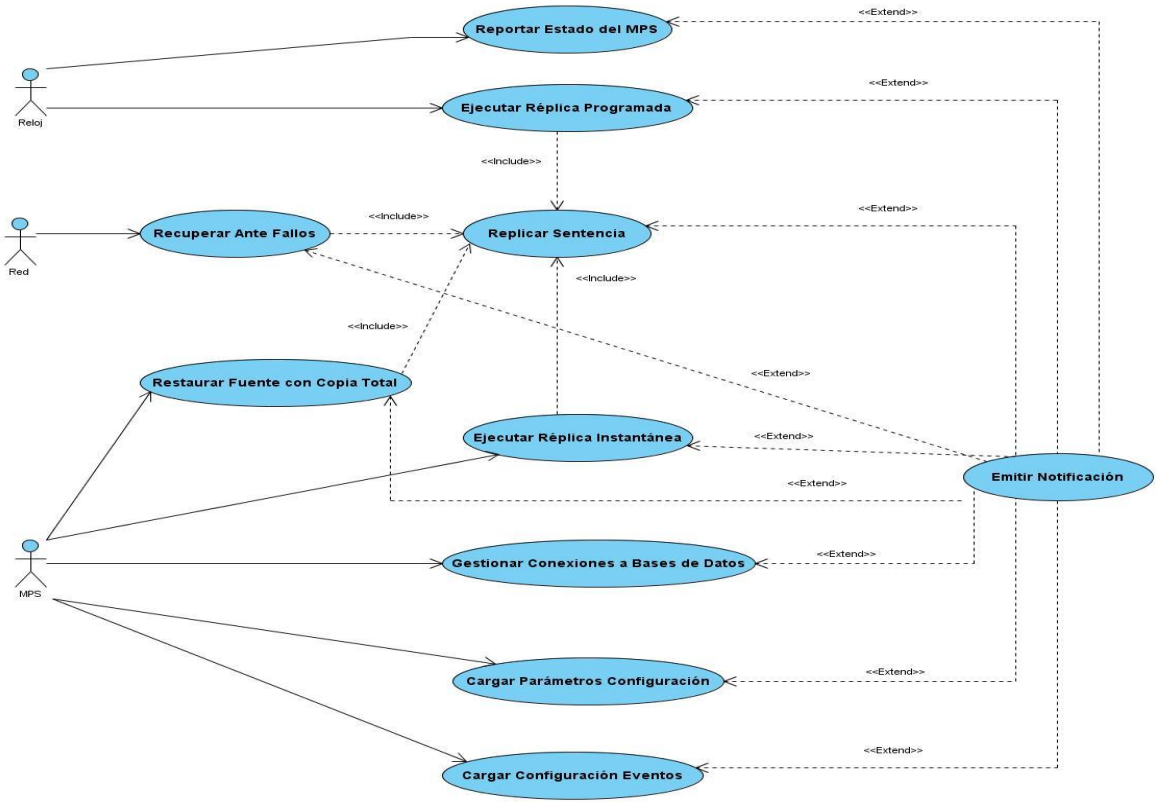


Figura 2. Diagrama de Casos de Uso.

2.7.4 Descripción de los casos de uso

Tabla 3. Descripción del CU Ejecutar réplica instantánea

Caso de uso	
CU-1	Ejecutar réplica instantánea
Propósito	Replicar los datos de fuentes orígenes a fuentes destinos, según la configuración de eventos instantáneos.
Actores : mps	
Resumen	El caso de uso se inicia cuando se modifica una fuente de datos y la librería libtransaccionesql.so notifica al MPS que ha ocurrido un cambio en una fuente de datos.

Referencias	RF1,RF1.1,RF1.2,RF1.3
Acción del actor 1. El mps inicia la réplica instantánea	**Respuesta del sistema** 2. El sistema solicita la configuración de eventos instantáneos. 3. El sistema busca la información de las bases de datos orígenes que intervienen en esa sincronización. 4. El sistema crea un hilo de ejecución para cada una de las bases de datos orígenes. 4.1 Si no puede crearse un hilo de ejecución ir a Sección Notificar Mensaje. 5. El sistema crea un Nodo de Trabajo de Réplica con la información de las bases de datos y se asignan a cada uno de los hilos de ejecución. 6. El sistema ejecuta los procedimientos asociados a los hilos de réplica.
Flujo alterno 4.1 Sección Notificar mensaje	
Acción del actor	**Respuesta del sistema** 4.1.1 El sistema envía una notificación a la Base de Datos origen con el mensaje “Imposible crear hilo de ejecución”.
Puntos de extensión.	
En el paso 5 Ejecutar Procedimientos asociados a los hilos de réplica. Ver CU 3 Replicar Sentencia. En el paso 4.1.1 Notificar mensaje a Base de Datos. Ver CU 5 Emitir Notificación.	

2.8 Matriz de trazabilidad

En la siguiente tabla se muestra la trazabilidad de los requisitos funcionales y los casos de uso del sistema.

Tabla 4. Matriz de Trazabilidad

RF / CU	RF1	RF2	RF3	RF4	RF5	RF6	RF7	RF8	RF9	RF10
CU1	X									
CU2		X								
CU3	X	X	X							
CU4				X						

CU5					X					
CU6						X				
CU7							X			
CU8								X		
CU9									X	
CU10										X

2.9 Patrones a utilizar

Para la descripción y la modelación del diagrama de casos de uso del sistema vistos en este capítulo, se utilizaron los siguientes patrones de casos de uso para garantizar mayor claridad en las descripciones. Los patrones de casos de uso [24] que se utilizaron fueron:

Concordancia

Extrae una subsecuencia de acciones que aparecen en diferentes lugares del flujo de casos de uso y es expresado por separado.

- *Reusabilidad*

Consta de 3 casos de uso. El primero llamado subsecuencia común, modela una secuencia de acciones que aparecerán en múltiples casos de uso en el modelo. Los otros casos de uso modelan el uso del sistema que comparte la subsecuencia común de acciones. De manera que deben existir al menos dos de ellos.

- *Adición*

En el caso de este patrón alternativo, la subsecuencia común de casos de uso, extiende los casos de uso compartiendo la subsecuencia de acciones. Los otros casos de uso modelan el flujo que será expandido con la subsecuencia. Este patrón es preferible usarlo cuando otros casos de uso se encuentran propiamente completos, o sea, que no requieren de una subsecuencia común de acciones para modelar los usos completos del sistema.

CRUD (*Creating, Reading, Updating, Deleting*)

Este patrón se basa en la fusión de casos de uso simples para formar una unidad conceptual.

- *Completo*

Este patrón consta de un caso de uso, llamado Información CRUD o Gestionar información, modela todas las operaciones que pueden ser realizadas sobre una parte de la información de un tipo específico, tales como creación, lectura, actualización y eliminación. Suele ser utilizado cuando todos los flujos contribuyen al mismo valor del negocio, y estos a su vez son cortos y simples.

Conclusiones

En este capítulo se describió el sistema de la propuesta de solución a partir del estudio realizado de la situación problemática. Se definieron los requisitos que debe cumplir el sistema en toda su totalidad, así como la definición de los casos de uso, incluyendo la descripción de cada una de sus funcionalidades. Partiendo de todas las características que se han expuesto, están las condiciones creadas para realizar el diseño de la solución propuesta.

Capítulo 3. Diseño del sistema

Introducción

En el presente capítulo se describen los pasos que se llevaron a cabo en el diseño del Motor de Procesamiento de Sentencias del sistema Chronos, siguiendo específicamente los pasos establecidos en la metodología de desarrollo seleccionada, lo que será de gran ayuda para la comprensión de la solución propuesta. Se caracterizan los patrones utilizados en el diseño de la descripción de las clases.

3.1 Diseño

En el diseño se modela el sistema, contribuyendo a una arquitectura estable y sólida, para que soporte todos los requisitos, tanto funcionales como no funcionales. El diseño posibilita una entrada apropiada y un punto de partida para las actividades de la implementación, descompone los trabajos de implementación en partes más manejables que pueden ser realizados por diferentes equipos de desarrollo. [25]

3.2 Diagrama de clases

Este diagrama de clases representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas.

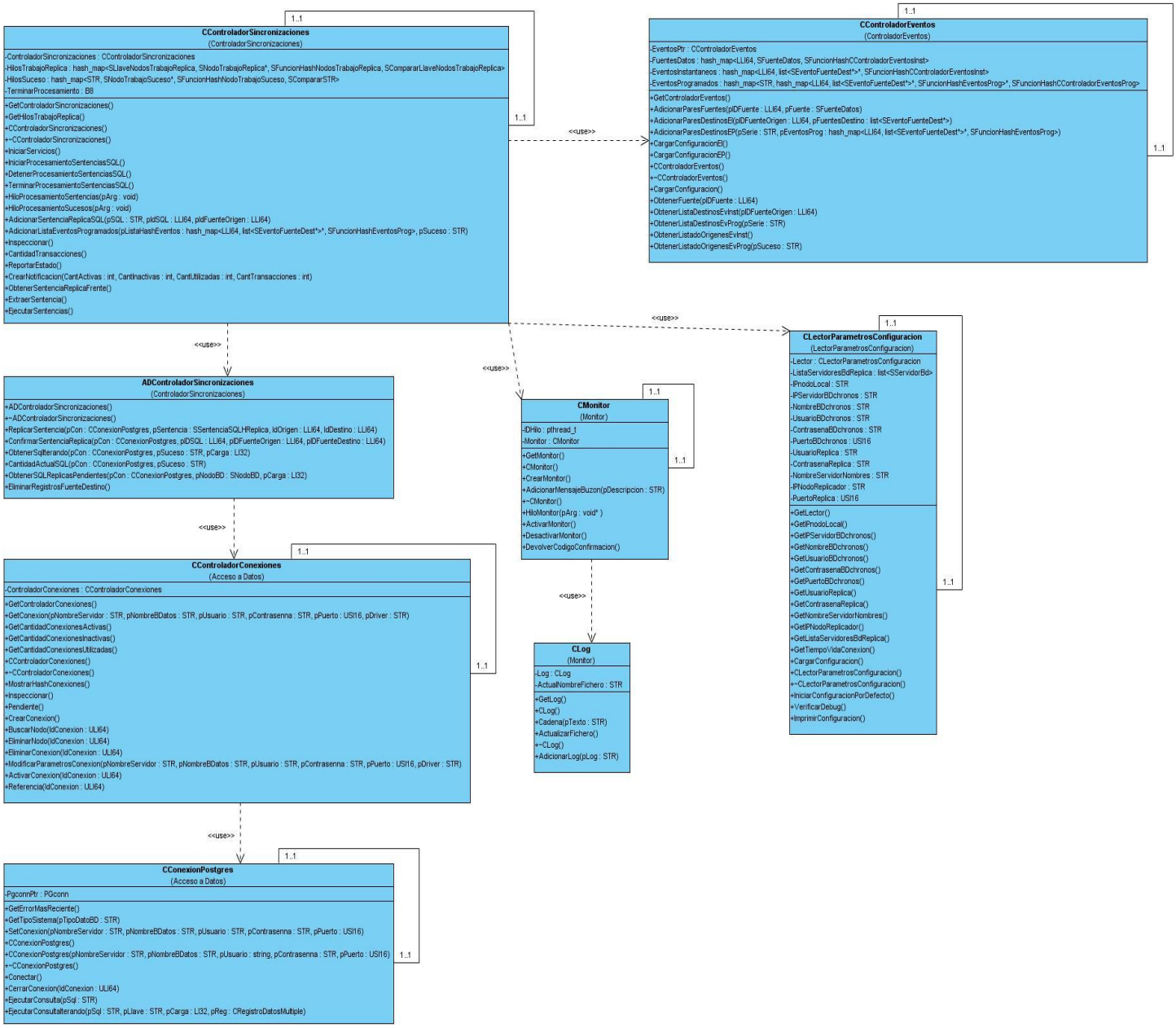


Figura 3 Diagrama de Clases del Diseño CU Ejecutar Réplica Instantánea.

Descripción de las clases.

Tabla 5. Descripción de clase CConexionPostgres

Nombre: CConexionPostgres	
Tipo de clase: controladora	
Atributo	**Tipo**
PgconnPtr	PGconn
Para cada responsabilidad:	
Nombre:	**CConexionPostgres()**
Descripción:	Constructor de la clase sin parámetros. Asigna valores por defecto a los parámetros de conexión.
Nombre:	**CConexionPostgres**(STR pNombreServidor, STR pNombreBDatos, string pUsuario, STR pContrasenna, USI16 pPuerto)
Descripción:	Constructor de la clase con parámetros. Establece los parámetros de conexión a la BD. Hace la conexión y verifica el estado de la misma, en caso de error lanza una excepción de tipo EBDImpConexion.
Nombre:	virtual ~**CConexionPostgres()**
Descripción:	Destructor de la clase CConexionPostgres. Cierra la conexión existente.
Nombre:	void **Conectar()**
Descripción:	Permite la conexión a la base de datos dado los parámetros de conexión.
Nombre:	void **CerrarConexion**(ULI64 IdConexion)
Descripción:	Cierra la conexión existente.
Nombre:	void **SetConexion**(STR pNombreServidor,STR pNombreBDatos, STR pUsuario,STR pContrasenna, USI16 pPuerto)
Descripción:	Cambia los parámetros de conexión y restablece la conexión con nuevos parámetros.
Nombre:	STR **GetErrorMasReciente()**
Descripción:	Devuelve el mensaje de error más reciente de esta conexión a la Base de datos.
Nombre:	CRegistroDatosMultiple **EjecutarConsulta**(STR pSql)
Descripción:	Ejecutar una consulta a la base de datos. Se devuelve la estructura RegistroDatosMultiple con el resultado.
Nombre:	CRegistroDatosMultiple **EjecutarConsultaIterando**(STR pSql, STR pLlave, LI32 pCarga, CRegistroDatosMultiple* pReg)
Descripción:	Ejecutar una consulta a la base de datos iterando, regulando la carga. Se devuelve la estructura RegistroDatosMultiple con el resultado.
Nombre:	TIPODATO **GetTipoSistema**(STR pTipoDatoBD)

Descripción:	A partir del tipo de dato postgres devuelve el tipo de datos del sistema.

Tabla 6. Descripción de clase CControladorConexiones

Nombre: CControladorConexiones	
Tipo de clase: controladora	
Atributo	**Tipo**
ControladorConexiones	CControladorConexiones
Conexiones	hash_map<SNodoBD, IConexion*, SFuncionHashConexiones, SCompararNodoBD>
Para cada responsabilidad:	
Nombre:	**CControladorConexiones()**
Descripción:	Constructor de la clase. Inicializa los valores por defecto.
Nombre:	virtual **~CControladorConexiones()**
Descripción:	Destructor de la clase. Borra todas las conexiones del hash y elimina la estructura. Esto invoca el constructor de CConexionPostgres que se encarga de cerrar las conexiones que estén activas en ese momento.
Nombre:	void **GetControladorConexiones()**
Descripción:	Implementación del Singleton para CControladorConexiones. Garantiza que el CControladorConexiones sea único.
Nombre:	void **GetConexion**(STR pNombreServidor, STR pNombreBDatos, STR pUsuario, STR pContrasenna, USI16 pPuerto, STR pDriver)
Descripción:	Retorna una CConexionPostgres, con los parámetros de conexión brindados. Propaga el Error EImpConexion en caso de que no pueda establecerse.
Nombre:	void **MostrarHashConexiones()**
Descripción:	Imprime la información relacionada a las conexiones activas.
Nombre:	void **Inspeccionar()**
Descripción:	Inspecciona las conexiones, verificando el estado de las mismas y toma decisiones al respecto.
Nombre:	int **GetCantidadConexionesActivas()**
Descripción:	Devuelve la cantidad de conexiones activas que existen en el momento.
Nombre:	int **GetCantidadConexionesInactivas()**
Descripción:	Devuelve la cantidad de conexiones inactivas que existen en el momento.
Nombre:	int **GetCantidadConexionesUtilizadas()**

Descripción:	Devuelve la cantidad de conexiones que se están utilizando.
Nombre:	void **Pendiente()**
Descripción:	Mantiene un nodo en estado pendiente.
Nombre:	void **CrearConexion()**
Descripción:	Crea una nueva conexión.
Nombre:	SNodo **BuscarNodo**(ULI64 IdConexion)
Descripción:	Busca un nodo dado su id.
Nombre:	void **EliminarNodo**(ULI64 IdConexion)
Descripción:	Elimina un nodo dado su id .
Nombre:	void **EliminarConexion**(ULI64 IdConexion)
Descripción:	Elimina la conexión dado su id.
Nombre:	void **ModificarParametrosConexion**(STR pNombreServidor, STR pNombreBDatos, STR pUsuario, STR pContrasenna, USI16 pPuerto, STR pDriver)
Descripción:	Modifica los parámetros de conexión.
Nombre:	void **ActivarConexion**(ULI64 IdConexion)
Descripción:	Activa una conexión dado su id.
Nombre:	void **Referencia**(ULI64 IdConexion)
Descripción:	Devuelve una referencia a la conexión establecida.

Tabla 7. Descripción de clase CControladorEventos

Nombre: CControladorEventos	
Tipo de clase: controladora	
Atributo	**Tipo**
EventosPtr	CControladorEventos
FuentesDatos	hash_map<LLI64,SFuenteDatos,SFuncionHashCControladorEventosInst>
EventosInstantaneos	hash_map<LLI64,list<SEventoFuenteDest*>*,SFuncionHashCControladorEventosInst>
EventosProgramados	hash_map<STR, hash_map< LLI64,list<SEventoFuenteDest*>* ,SFuncionHashEventosProg>*,SFuncionHashCControladorEventosProg>
Para cada responsabilidad:	

Nombre:	void **AdicionarParesFuentes**(LLI64 pIDFuente, SFuenteDatos pFuente)
Descripción:	Método privado que adiciona los pares (origen, fuente) al hash_map FuentesOrigen. Usado por CargarConfiguracion.
Nombre:	void **AdicionarParesDestinosEI**(LLI64 pIDFuenteOrigen, list<SEventoFuenteDest*>* pFuentesDestino)
Descripción:	Método privado que adiciona los pares (origen, listadestinos) al hash_map Eventos Instantáneos. Usado por CargarConfiguracion.
Nombre:	void **AdicionarParesDestinosEP**(STR pSerie, hash_map<LLI64,list<SEventoFuenteDest*> *,SFuncionHashEventosProg> * pEventosProg)
Descripción:	Método privado que adiciona los pares (origen, listadestinos) al hash_map Eventos Programados. Usado por Cargar configuración.
Nombre:	void **CargarConfiguracionEI()**
Descripción:	Método privado que carga la configuración de Eventos Instantáneos.
Nombre:	void **CargarConfiguracionEP()**
Descripción:	Método privado que carga la configuración de Eventos Programados.
Nombre:	**CControladorEventos()**
Descripción:	Constructor de la clase.
Nombre:	STR **GetControladorEventos()**
Descripción:	Devuelve la única instancia de la clase, patrón Singleton.
Nombre:	virtual **~CControladorEventos()**
Descripción:	Destructor de la clase.
Nombre:	void **CargarConfiguracion()**
Descripción:	Carga la configuración de Eventos Instantáneos y Programados.
Nombre:	void **RecargarConfiguracion()**
Descripción:	Borra la configuración existente de Eventos de sincronización y la recarga nuevamente.
Nombre:	SFuenteDatos **ObtenerFuente**(LLI64 pIDFuente)
Descripción:	Devuelve los datos de una fuente incluyendo los parámetros de conexión a partir de su identificador.
Nombre:	list<SEventoFuenteDest>**ObtenerListaDestinosEvInst**(LLI64 pIDFuenteOrigen)
Descripción:	Devuelve la lista de destinos asociados a un idorigen de un Evento Instantáneo.
Nombre:	hash_map<LLI64, list<SEventoFuenteDest>, SFuncionHashEventosProg> **ObtenerListaDestinosEvProg**(STR pSerie)

Descripción:	Devuelve el hash map asociado a los eventos programados, dada una serie.
Nombre:	list<SEventoFuenteDest> **ObtenerListadoOrigenesEvInst()**
Descripción:	Devuelve la lista de orígenes de un evento instantáneo.
Nombre:	list<SEventoFuenteDest> **ObtenerListadoOrigenesEvProg**(STR Suceso)
Descripción:	Devuelve la lista de orígenes asociados a un suceso.

Tabla 8. Descripción de clase ADControladorSincronizaciones

Nombre: ADControladorSincronizaciones	
Tipo de clase: controladora	
Para cada responsabilidad:	
Nombre:	**ADControladorSincronizaciones()**
Descripción:	Constructor de la clase.
Nombre:	virtual ~**ADControladorSincronizaciones()**
Descripción:	Destructor de la clase.
Nombre:	static void **ReplicarSentencia**(CConexionPostgres* pCon, SSentenciaSQLHReplica pSentencia)
Descripción:	Llamada al procedimiento almacenado insertar_tsql_replica.
Nombre:	static bool **ConfirmarSentenciaReplica**(CConexionPostgres* pCon, LLI64 pIDSQL, LLI64 pIDFuenteOrigen, LLI64 pIDFuenteDestino)
Descripción:	Llamada al procedimiento almacenado confirma_replica.
Nombre:	CRegistroDatosMultiple **ObtenerSqlIterando**(CConexionPostgres* pCon, STR pSuceso, LI32 pCarga)
Descripción:	Devuelve los registros de tsql listos para la replica de forma iterativa.
Nombre:	int **CantidadActualSQL**(CConexionPostgres* pCon, STR pSuceso)
Descripción:	Devuelve la cantidad actual de sentencias Sql almacenadas en tsql que corresponden al Suceso pSuceso.
Nombre:	CRegistroDatosMultiple **ObtenerSQLReplicasPendientes**(CConexionPostgres* pCon, SNodoBD pNodoBD, LI32 pCarga)
Descripción:	Devuelve los registros de tsql pendientes para la replica de forma iterativa.

Tabla 9. Descripción de clase CControladorSincronizaciones

Nombre: CControladorSincronizaciones
Tipo de clase: controladora

Atributo	**Tipo**
ControladorSincronizaciones	CControladorSincronizaciones
HilosTrabajoReplica	hash_map<SLlaveNodosTrabajoReplica, SNodoTrabajoReplica*, SFuncionHashNodosTrabajoReplica, SCompararLlaveNodosTrabajoReplica>
HilosSuceso	hash_map<STR, SNodoTrabajoSuceso*, SFuncionHashNodoTrabajoSuceso, SCompararSTR>
TerminarProcesamiento	B8

Para cada responsabilidad:	
Nombre:	**CControladorSincronizaciones()**
Descripción:	Constructor de la clase
Nombre:	virtual **~CControladorSincronizaciones()**
Descripción:	Destructor de la clase.
Nombre:	void **GetControladorSincronizaciones()**
Descripción:	Aplicación del Patrón singleton. Devuelve el Ptr del Controlador de Sincronizaciones, garantizando que este sea único
Nombre:	void **IniciarServicios()**
Descripción:	Inicia los servicios del Controlador de Sincronizaciones.
Nombre:	void **IniciarProcesamientoSentenciasSQL()**
Descripción:	Inicia el procesamiento de sentencias SQL del hilo de procesamiento de Sentencias.
Nombre:	void **DetenerProcesamientoSentenciasSQL()**
Descripción:	Detiene el Hilo de procesamiento de sentencias SQL.
Nombre:	void **TerminarProcesamientoSentenciasSQL()**
Descripción:	Termina el procesamiento del Hilo de Sentencias SQL.
Nombre:	void **GetHilosTrabajoReplica()**
Descripción:	Devuelve la estructura que almacena los hilos de réplica instantánea existentes. (hash_map)
Nombre:	static void **HiloProcesamientoSentencias**(void *pArg)
Descripción:	Implementación de la Función del Hilo de Procesamiento de Sentencias.
Nombre:	static void **HiloProcesamientoSucesos**(void *pArg)
Descripción:	Implementación de la función del Hilo de Procesamiento de Sucesos.
Nombre:	void **AdicionarSentenciaReplicaSQL**(STR pSQL, LLI64 pIdSQL, LLI64

	pIdFuenteOrigen)
Descripción:	Adiciona Sentencias de Replica a la tabla tsql, listas para replicarse.
Nombre:	void **AdicionarListaEventosProgramados**(hash_map<LLI64, list<SEventoFuenteDest*>*, SFuncionHashEventosProg> *pListaHashEventos, STR pSuceso)
Descripción:	Adiciona una lista de eventos programados asociados a un Suceso al hash de Eventos programados.
Nombre:	void **Inspeccionar()**
Descripción:	Revisa si las conexiones están activas o no.
Nombre:	STR **ReportarEstado()**
Descripción:	Crea y envía un reporte del estado del MPS.
Nombre:	void **CrearNotificacion**(int CantActivas, int CantInactivas, int Contabilizadas, inti CantTransacciones)
Descripción:	Crea una notificación con los parámetros indicados.
Nombre:	STR **ObtenerSentenciaReplicaFrente()**
Descripción:	Obtiene la sentencia de replica emitida en ese instante.
Nombre:	void **ExtraerSentencia()**
Descripción:	Elimina una sentencia de replica de la Base de Datos.
Nombre:	int **CantidadTransacciones()**
Descripción:	Devuelve la cantidad de transacciones que se están realizando.

Tabla 10. Descripción de clase CLectorParametrosConfiguracion

Nombre: CLectorParametrosConfiguracion	
Tipo de clase: controladora	
Atributo	**Tipo**
Lector	CLectorParametrosConfiguracion
ListaServidoresBdReplica	list<SServidorBd>
IPnodoLocal	STR
IPServidorBDchronos	STR
NombreBDchronos	STR
UsuarioBDchronos	STR
ContrasenaBDchronos	STR
PuertoBDchronos	USI16

UsuarioReplica	STR
ContrasenaReplica	STR
NombreServidorNombres	STR
IPNodoReplicador	STR
PuertoReplica	USI16
TiempoVidaConexion	USI16
TiempoTransaccion	USI16
Debug	USI16
ValorDebug	ULLI64
DirCarpetaLog	STR
ValorLog	ULLI64
Para cada responsabilidad:	
Nombre:	void **CargarConfiguracion()**
Descripción:	Carga la configuración desde un fichero.
Nombre:	**CLectorParametrosConfiguracion()**
Descripción:	Constructor de la clase.
Nombre:	virtual **~CLectorParametrosConfiguracion()**
Descripción:	Destructor de la clase.
Nombre:	CLectorParametrosConfiguracion **GetLector()**
Descripción:	Devuelve la única instancia de la clase.
Nombre:	STR **GetIPnodoLocal()**
Descripción:	Devuelve el IP del nodo local.
Nombre:	STR **GetIPServidorBDchronos()**
Descripción:	Devuelve el IP del Servidor de la Base de datos Chronos.
Nombre:	STR **GetNombreBDchronos()**
Descripción:	Devuelve el nombre de la Base de datos Chronos.
Nombre:	STR **GetUsuarioBDchronos()**
Descripción:	Devuelve el usuario de la Base de datos Chronos.
Nombre:	STR **GetContrasenaBDchronos()**
Descripción:	Devuelve la contraseña de la Base de datos Chronos.
Nombre:	TIPODATO **GetPuertoBDchronos()**
Descripción:	Devuelve el puerto de la Base de datos Chronos.
Nombre:	STR **GetUsuarioReplica()**

Descripción:	Devuelve el usuario de la replica.
Nombre:	STR **GetContrasenaReplica()**
Descripción:	Devuelve la contraseña de la replica.
Nombre:	STR **GetNombreServidorNombres()**
Descripción:	Devuelve el nombre del Servidor de Nombres.
Nombre:	STR **GetIPNodoReplicador()**
Descripción:	Devuelve el IP del nodo replicador.
Nombre:	list<SNodoBD> **GetListaServidoresBdReplica()**
Descripción:	Devuelve una lista con los servidores de Base de datos de replica.
Nombre:	TIPODATO **GetTiempoVidaConexion()**
Descripción:	Devuelve el tiempo de vida de conexión.
Nombre:	TIPODATO **GetDebug()**
Descripción:	Devuelve el valor del nivel de debug en que se está ejecutando la instancia del MPS.
Nombre:	void **IniciarConfiguracionPorDefecto()**
Descripción:	Carga los parámetros por defecto que están definidos en el sistema.
Nombre:	void **VerificarDebug()**
Descripción:	Verifica el valor del parámetro Debug.
Nombre:	void **ImprimirConfiguracion()**
Descripción:	Si el parámetro Debug ==1 imprime la configuración.

Tabla 11. Descripción de clase CLog

Nombre: CLog	
Tipo de clase: controladora	
Atributo	**Tipo**
Log	CLog
ActualNombreFichero	STR
Para cada responsabilidad:	
Nombre:	**CLog()**
Descripción:	Constructor de la clase.
Nombre:	STR **Cadena**(STR pTexto)
Descripción:	Devuelve una cadena STR concatenando el mensaje a la fecha actual en se lanza.
Nombre:	void **ActualizarFichero()**
Descripción:	Establece el nombre del fichero.

Nombre:	virtual **~CLog()**
Descripción:	Destructor de la clase.
Nombre:	static CLog **GetLog()**
Descripción:	Devuelve la instancia única de la clase, patrón Singleton.
Nombre:	void **AdicionarLog(**STR pLog**)**
Descripción:	Adiciona al final del fichero el nuevo Log.

Tabla 12. Descripción de clase CMonitor

Nombre: CMonitor	
Tipo de clase: controladora	
Atributo	**Tipo**
IDHilo	pthread_t
Monitor	CMonitor
Para cada responsabilidad:	
Nombre:	**CMonitor()**
Descripción:	Constructor sin parámetros de la clase.
Nombre:	CMonitor **GetMonitor()**
Descripción:	Devuelve una instancia única de la clase CMonitor.
Nombre:	void **AdicionarMensajeBuzon**(TIPOMENSAJE pTipoMensaje, STR pDescripcion)
Descripción:	Adiciona mensajes a la Base de Datos.
Nombre:	virtual **~CMonitor()**
Descripción:	Destructor de la clase.
Nombre:	void **HiloMonitor**(void* pArg)
Descripción:	Función asociada al hilo de recopilación de información del Monitor.
Nombre:	void **ActivarMonitor()**
Descripción:	Inicia el proceso de recopilación de información del Monitor.
Nombre:	void **DesactivarMonitor()**
Descripción:	Detiene el proceso de recopilación de información del Monitor.

Las definiciones enumerativas para TIPODATO y MODO se refieren a los valores de los tipos de datos soportados por el sistema y los modos de inserción de los registros en las clases persistentes respectivamente.

Las funciones marcadas (*) se describen de manera genérica, puesto que se repite en la clase correspondiente la misma funcionalidad para cada tipo de dato del sistema representado por X.

Las estructuras de almacenamiento de datos utilizadas se corresponden con las estructuras de STL de C++, y las que se utilizan son queue, hash_map y list.

3.3 Tipos de datos soportados.

Tabla 13. Descripción de los tipos de datos soportados por el Motor de Procesamiento de Sentencias.

Tipo	Descripción
C8	Caracter con signo de 8 bits.
UC8	Caracter sin signo de 8 bits.
I32	Entero con signo de 32 bits.
UI32	Entero sin signo de 32 bits.
LI32	Entero largo con signo de 32 bits.
ULI32	Entero largo sin signo de 32 bits.
LLI64	Entero con signo de 64 bits.
SI16	Entero corto con signo de 16 bits.
USI16	Entero corto sin signo de 16 bits.
F32	Flotante de precisión simple de 32 bits.
D64	Flotante de precisión doble de 64 bits.
LD96	Flotante de precisión doble de 96 bits.
STR	Cadena de caracteres.
B8	Booleano de 8 bits.
UUID	Arreglo de caracteres sin signo de 128 bits.

3.4 Diagramas de interacción

Los diagramas de interacción explican gráficamente cómo los objetos interactúan a través de mensajes para realizar las tareas. Dentro de estos diagramas están los de secuencia y los de colaboración. Para

representar gráficamente ésta interacción se escogió el diagrama de secuencia para alcanzar mayor claridad.

3.4.1 Diagramas de Secuencia

Este diagrama de secuencia destaca la ordenación temporal de los mensajes (mensaje simple, síncrono, asíncrono, y/o de retorno) en la realización de los casos de uso.

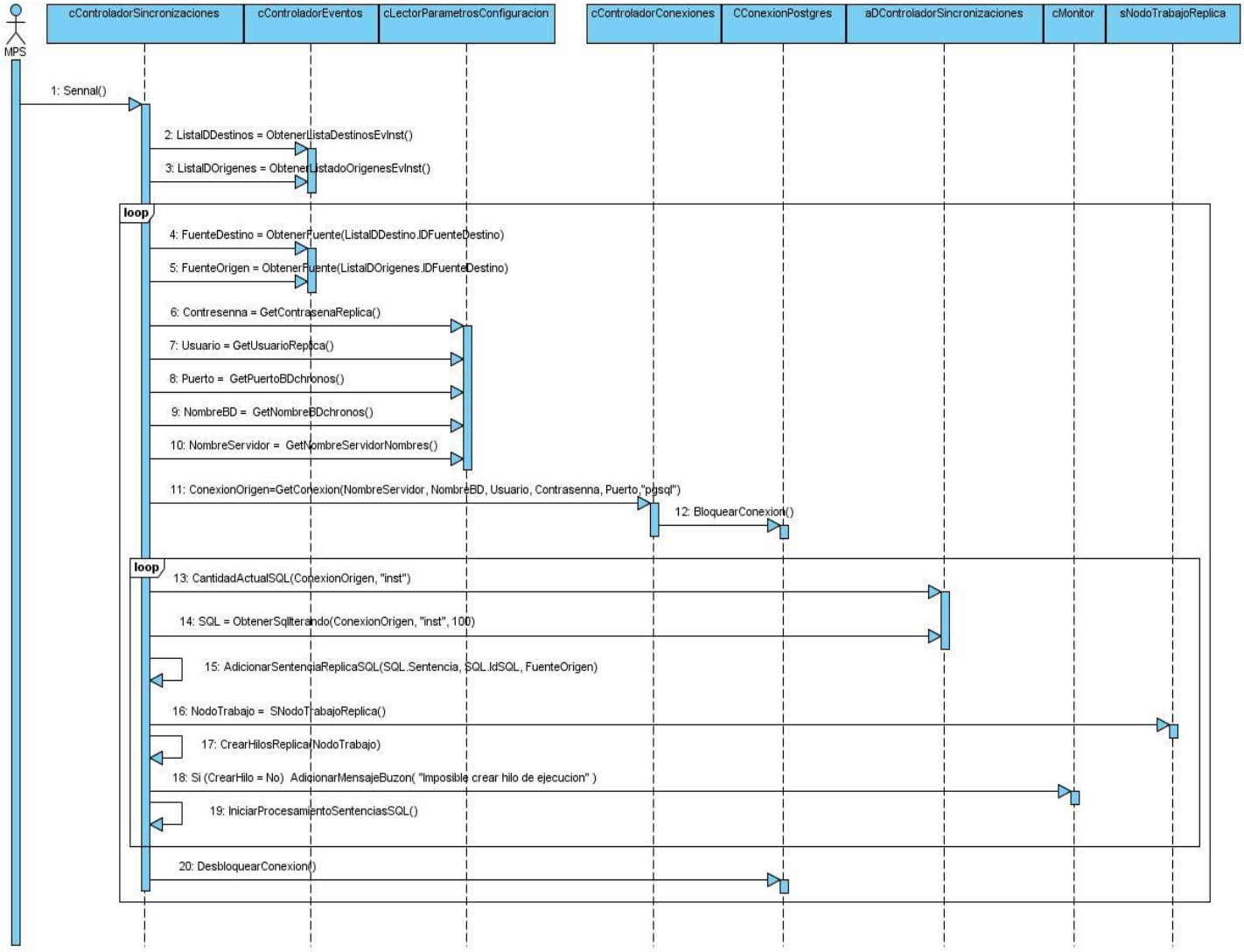


Figura 4 Diagrama de Secuencia CU Ejecutar Réplica Instantánea.

- En la modelación de los diagramas de secuencia se utilizaron los estándares de código para C++ mostrados en el **Anexo # 2**.

- Modelo de datos

Este modelo de datos es usado para describir la representación lógica y física de la información persistente manejada por el sistema.

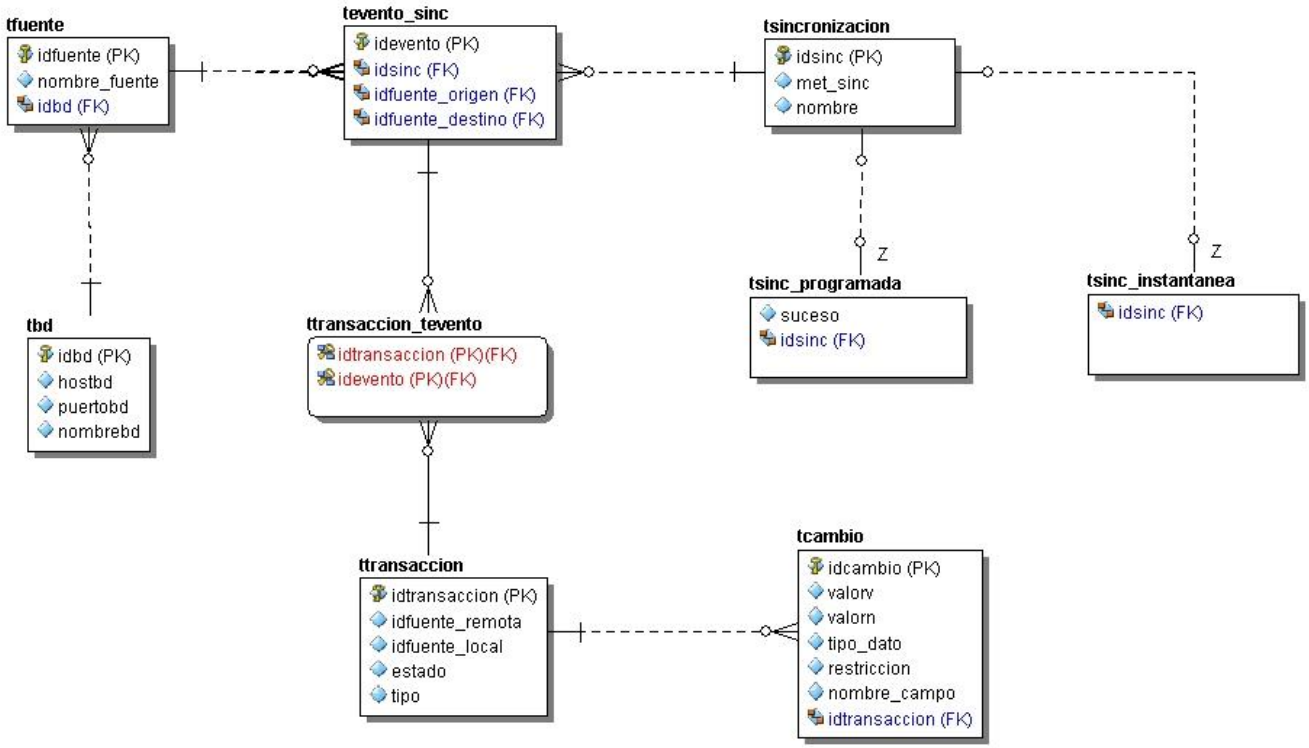


Figura 5 Diagrama Entidad Relación

En la modelación del diagrama entidad relación se aplicaron estándares de diseño de Base de Datos mostrados en el **Anexo # 1.**

3.5 Descripción de las tablas.

Tabla 14 Descripción de la tabla rttransaccion_tevento

Nombre: rttransaccion_tevento		
Descripción: Representa la relación M...M entre la tabla ttransacción y tevento.		
Atributo	**Tipo**	**Descripción**
Idsql	integer	Identificador de la sentencia SQL.
idevento	integer	Identificador del evento.

Tabla15 Descripción de la tabla tbd

Nombre: tbd		
Descripción: Almacena la información de las bases de datos que interactúan en la réplica con la base de datos en la que se encuentra el esquema chronos.		
Atributo	**Tipo**	**Descripción**
idbd	integer	Identificador de la Base de Datos.
nombrebd	text	Nombre de la Base de Datos.
hostbd	text	Host de la Base de Datos.
puertobd	smallint	Se almacena el puerto de la Base de Datos.

Tabla 16 Descripción de la tabla tevento_sinc

Nombre: tevento_sinc		
Descripción: Almacena todos los eventos de sincronización configurados. Por cada evento se almacena la fuente origen, la fuente destino y el tipo de sincronización.		
Atributo	**Tipo**	**Descripción**
idevento	integer	Identificador del evento.
idfuente_origen	integer	Identificador de la fuente origen.
idfuente_destino	integer	Identificador de la fuente destino.
idsinc	integer	Identificador de sincronización.

Tabla 17 Descripción de la tabla tfuente

Nombre: tfuente		
Descripción: Almacena todas las fuentes de datos involucradas en la réplica del nodo en el que se encuentra el esquema chronos.		
Atributo	**Tipo**	**Descripción**
idfuente	bigint	Identificador de la fuente.
nombre_fuente	text	Nombre de la fuente.
idbd	integer	Identificador de la Base de Datos.

Tabla 18 Descripción de la tabla tsinc_instantanea

Nombre: tsinc_instantanea		
Descripción: Almacena la información de las sincronizaciones que son instantáneas.		
Atributo	**Tipo**	**Descripción**
idsinc	integer	Identificador de la sincronización.

Tabla 19 Descripción de la tabla tsinc_programada

Nombre: tsinc_programada		
Descripción: Almacena la información de las sincronizaciones que son programadas.		
Atributo	**Tipo**	**Descripción**
idsinc	integer	Identificador de la sincronización.
suceso	character	Secuencia de caracteres que representa el momento en que se dispara la réplica programada.

Tabla 20 Descripción de la tabla tsinconizacion

Nombre: tsincronizacion		
Descripción: Almacena la información de las sincronizaciones que se configuran en el sistema.		
Atributo	**Tipo**	**Descripción**
idsinc	integer	Identificador de la sincronización.
nombre	text	Nombre de la sincronización.
met_sinc	integer	Método de sincronización.

Tabla 21 Descripción de la tabla ttransaccion

Nombre: ttransaccion		
Descripción: Amacena la información de las transacciones durante la réplica, desde que se capturan hasta que se replican.		
Atributo	**Tipo**	**Descripción**
idtransaccion	bigint	Identificador de la transacción.
estado	character	Estado de la transacción.
tipo	character	Tipo de transacción.
idfuente_remota	bigint	Identificador de la fuente remota.
idfuente_local	bigint	Identificador de la fuente local.

Tabla 22 Descripción de la tabla tcambio

Nombre: tcambio		
Descripción: Amacena la información de las transacciones durante la réplica, desde que se capturan hasta que se replican.		
Atributo	**Tipo**	**Descripción**
idcambio	bigint	Identificador del cambio.
idtransaccion	bigint	Identificador de la transacción.
nombre_campo	character	Nombre del campo.
valorn	character	Valor nuevo del campo.
valorv	character	Valor nuevo del campo.
tipo_dato	character	Tipo de dato.
restriccion	character	Restricción del campo: PK si es llave primaria, FK si es llave extranjera, U si es valor único o NR si no tiene restricción.

- Definiciones de diseño que se aplican.

En el diseño de la Base de Datos se definió un esquema (chronos). El esquema chronos contiene: tablas, funciones, disparadores y secuencias.

Dentro del esquema chronos se definieron 4 funciones, utilizando el lenguaje plpgsql:

- chronos.confirma_replica (bigint, bigint, bigint): Confirma la realización correcta de la réplica de una transacción, eliminando sus datos asociados una vez que esta se replicó correctamente.

- chronos.fregistrar_cambios_uaplicacion (bigint, integer, character varying, variadic text[], variadic text[], variadic text[], variadic text[]): Utilizada por la líbrería libtransacciones.so cuando el usuario que replica es externo. Actualiza los cambios en la tabla tcambio que se realizan en una fuente de datos origen realizados por el usuario de aplicación. Actualiza los eventos asociados a la transacción en la tabla rttransaccion_tevento e inserta la transacción nueva en la tabla ttransaccion.

- chronos.fregistrar_cambios_ureplica (integer, variadic text [], variadic text [], variadic text [], variadic text []): Utilizada por la líbrería libtransacciones.so cuando el usuario que replica es chronos. Actualiza los cambios en la tabla tcambio que se realizan en una fuente de datos origen realizados por el usuario de aplicación para la última transacción realizada.

- chronos.finsertar_sql_replica (text, text, bigint, bigint) Utilizada por el MPS en la réplica de réplicas. Actualiza los eventos asociados a la transacción en la tabla rttransaccion_tevento e inserta la transacción nueva en la tabla ttransacción. Bloquea la tabla ttransacion para que fregistrar_cambios_ureplica pueda actualizar los cambios para esta operación.

Se utiliza un disparador:

chronos.ftriggertaplicacion (): Función que realizan los disparadores asociados a las tablas de aplicación orígenes involucradas en la réplica. Es la encargada de establecer el vínculo con la librería libtransaccionesql.so.

Dentro de las secuencias se definieron:

- chronos.sec_nestado_transaccion_seq: Permite incrementar en uno el valor del campo X cada vez que se modifique el estado de la transacción en la tabla Y.

-chronos.sec_tcambio_seq: Permite incrementar en uno el valor del campo X cada vez que se ejecute un cambio en la tabla Y.

-chronos.sec_tevento_sinc_seq: Permite incrementar en uno el valor del campo X cada vez que se inserta un registro en la tabla Y-chronos.sec_transaccion_seq: Permite incrementar en uno cada vez que se ejecute una transacción.

Existen tablas como tbd, tfuente, tsincronizacion, instantánea y programada que no tienen secuencias asociadas porque el valor de sus llaves primarias es establecido por la aplicación de administración y configuración AdminChronos.

- Seguridad

Existen restricciones de llaves primarias y llaves foráneas asociadas para una selección adecuada de los tipos de datos.

Gran parte de la configuración es establecida por la aplicación de administración y configuración AdminChronos por lo que ninguna de las funciones asociadas pueden modificar los valores establecidos.

3.6 Descripción de la arquitectura propuesta

Arquitectura de 3 capas

Los sistemas o arquitecturas en capas definen cómo organizar el modelo de diseño a través de capas, que pueden estar físicamente distribuidas, lo cual quiere decir que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores. Es importante definir una arquitectura única sobre la cual construir la aplicación [26].

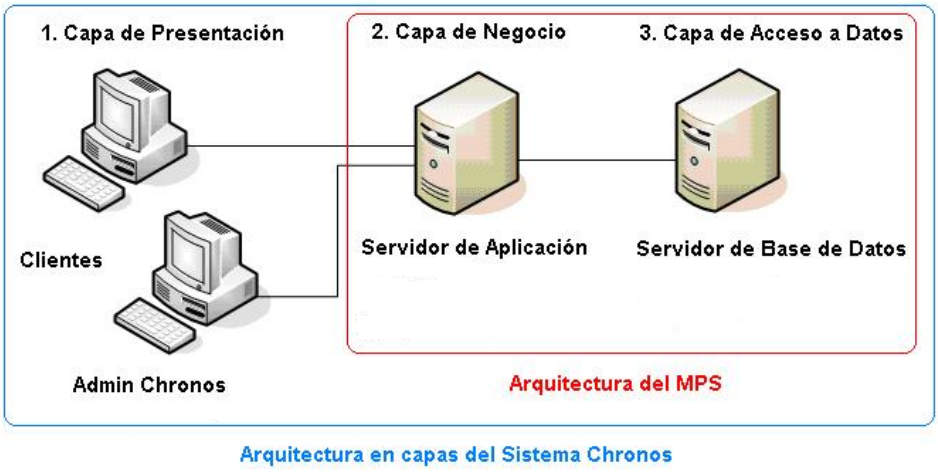


Figura 6. Arquitectura en capas

El sistema Chronos en general con todos sus componentes utiliza una arquitectura en 3 capas compuesta por la capa de presentación en donde se encuentra la interfaz de administración y configuración de la aplicación web AdminChronos, la capa de negocio que está compuesta por el Motor de Procesamiento de Sentencias (MPS) y el negocio de la aplicación AdminChronos y por la capa de Acceso a Datos que está compuesta por la Base de Datos y el esquema Chronos. El Motor de Procesamiento de Sentencias es un componente que forma parte del sistema Chronos y la arquitectura que presenta no posee la capa de presentación porque no tiene interfaz gráfica ni usuarios que interactúen con ella por lo cual solamente existe en la capa de Negocio y la capa de acceso a datos.

3.7 Patrones a utilizar

Para el diseño de las clases y los diagramas de interacción se emplearon algunos patrones GRASP [24] (*General Responsibility Assignment Software Patterns*, Patrones Generales de Software para Asignación de Responsabilidades) y GOF [27] con el propósito de lograr un buen diseño donde se pueda ver claramente las relaciones esenciales de los objetos.

Patrones de Asignación de Responsabilidades GRASP

Experto

La aplicación de este patrón permite a cada clase desarrollar las tareas que pueden realizar según la información que poseen.

Creador

Permite crear instancias de otras clases en correspondencia con la responsabilidad dada. Con esto se logró conservar el encapsulamiento ya que los objetos logran valerse de su propia información para realizar lo que se les pide.

Bajo acoplamiento

Este patrón soluciona el inconveniente de dar soporte a una dependencia escasa y a un aumento de la reutilización. El bajo acoplamiento estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento.

El bajo acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad.

Alta cohesión

Este patrón es utilizado para mantener la complejidad dentro de los límites manejables. Una clase con mucha cohesión es útil porque es fácil darle mantenimiento, entenderla y reutilizarla. Su alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, también simplifica el mantenimiento y los mejoramientos. El patrón alta cohesión presenta semejanzas con el mundo real. Se sabe que, si alguien asume demasiadas responsabilidades sobre todo las que debería delegar, no sería eficiente.

El diseño obtenido cumple con los patrones de Bajo acoplamiento y Alta cohesión permitiendo la colaboración entre los elementos del diseño, sin verse afectados la reutilización de los mismos y el entendimiento de estos cuando se encuentran aislados. La existencia de clases controladoras, como

CControladorEventos y CControladorSincronizaciones, evidencia la utilización de los patrones Creador y Experto.

Patrones según el Libro GOF (*Gang-Of-Four Book*)

Patrones Creacionales

Instancia única (*Singleton*): Este patrón garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia [28]

Patrones Estructurales

Fachada (*Facade*): El propósito de utilizar este patrón es proveer de una interfaz unificada y sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Este patrón permite que una biblioteca de software sea más fácil de usar y entender. Esto es posible porque la fachada implementa métodos convenientes para tareas comunes, puede reducir la dependencia de código externo en los trabajos internos, permitiendo así más flexibilidad en el desarrollo de sistemas.

Patrones de Comportamiento

Cadena de responsabilidad (*Chain of Responsibility*): Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.

Conclusiones

En el capítulo se muestran los artefactos generados en el flujo de trabajo de análisis y diseño, además se incluye una breve descripción de los mismos, con el objetivo de que se comprenda perfectamente los requisitos del software. Posibilitando la correcta transformación de los mismos a un diseño que indique como debe ser implementado el software. También se encuentra información referente a los patrones del diseño utilizados.

Capítulo 4. Validación del Diseño

Introducción

La medición es fundamental para cualquier disciplina de ingeniería, y en especial para la ingeniería del software. Las métricas son un buen medio para entender, monitorizar, controlar, predecir y probar el desarrollo de software. La medición permite tener una visión más profunda, proporcionando un mecanismo para la evaluación objetiva. Esta persigue tres objetivos fundamentales: ayudar a entender qué ocurre durante el desarrollo y el mantenimiento, permitir controlar qué es lo que ocurre en los proyectos y poder mejorar los procesos y los productos. En el presente capítulo se aplican un conjunto de métricas de diseño orientado a objeto y se analizan los resultados para determinar la calidad del diseño.

4.1 Tamaño de Clase (TC)

Para evaluar las métricas son necesarios los valores de los umbrales. Las medidas o umbrales para los parámetros de calidad han sido una polémica a nivel mundial en el diseño de sistemas. Algunos especialistas plantean umbrales para estas métricas según se muestra en la tabla 23, estos fueron los aplicados en el diseño de este sistema.

Tabla 23. Umbrales para TC**.**

Nro. De operaciones y/o atributos	
TC	Umbral
Pequeño	<= 20
Medio	>20 y <=30
Grande	>30

Al aplicar la métrica TC al diseño propuesto se obtuvieron los siguientes resultados: Para un total de 58 clases existentes en el diseño, se obtuvo un promedio de 1.3 atributos por clases y 4.3 operaciones, como se muestra en la tabla 24.

Tabla 24 Cantidad de clases de Diseño, operaciones y atributos promediados**.**

Total de clases	**Operaciones**	**Atributos**

58	4.3	1.3

De acuerdo con los umbrales propuestos en la tabla 23 se obtuvo que para un total de 58 clases que tiene el diseño, 54 son de tamaño pequeño, 2 son de tamaño medio y 2 son de tamaño grande como se muestra en la tabla 25, lo que representa el 3 % de clases medianas, el 94% de clases pequeñas y el 3% de clases grandes.

Tabla 25 Cantidad de clases por tamaño.

Cantidad de clases	Umbral	Tamaño
54	<= 20	Pequeño
2	>20 y <=30	Medio
2	>30	Grande

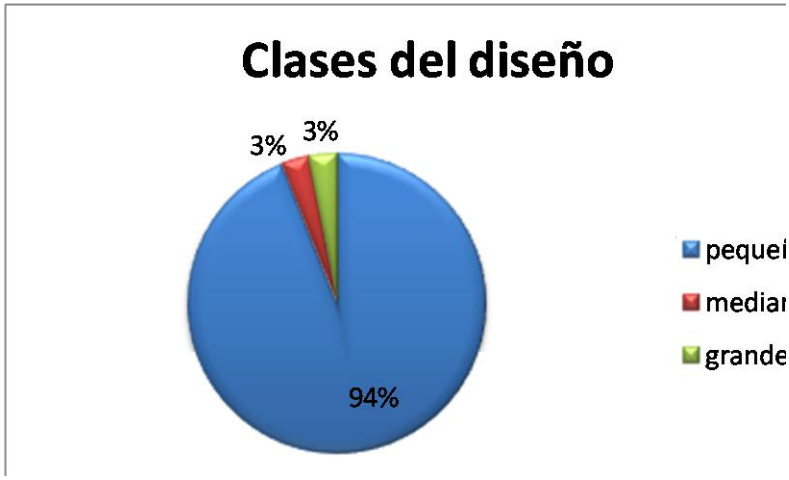


Figura 7 Representación de la Métrica TC

Estos valores demuestran que los indicadores de calidad reutilización, implementación y responsabilidad no se ven afectados.

4.2 Relaciones entre Clases (RC).

Tabla 26. Relaciones entre clases (RC).

Atributo que afecta	Modo en que lo afecta
Acoplamiento.	Un aumento del RC implica un aumento del Acoplamiento de la clase.
Complejidad de mantenimiento.	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
Reutilización.	Un aumento del RC implica una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Luego de efectuar los cálculos para un:

Total de clases	58
Promedio de asociaciones de uso	1,948275862

Se obtuvieron las siguientes gráficas:

Para el acoplamiento.

Tabla 27 Acoplamiento de clases

Acoplamiento	Cantidad de clases	Promedio
Ninguno	45	57,69230769
Bajo	3	3,846153846
Medio	2	2,564102564
Alto	8	10,25641026

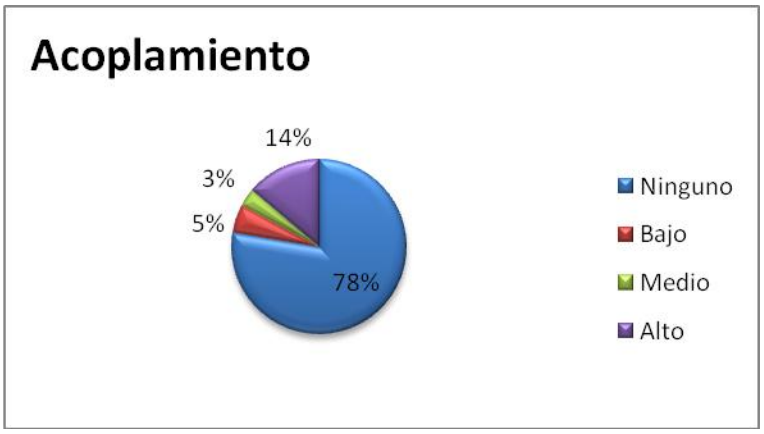


Figura 8 Representación del Acoplamiento

Para la Reutilización:

Tabla 28 Reutilización de clases.

Reutilización	Cantidad de clases	Promedio
Baja	8	10,25641026
Media	2	2,564102564
Alta	48	61,53846154

Figura 9 Representación de la Reutilización.

Para la complejidad de mantenimiento.

Tabla 29 Complejidad de mantenimiento de clases

Complejidad de Mantenimiento	Cantidad de clases	Promedio
Baja	48	61,53846154
Media	2	2,564102564
Alta	8	10,25641026

Figura 10 Representacion de la Complejidad de Mantenimiento.

Para la cantidad de Pruebas.

Tabla 30 Cantidad de Pruebas de clases.

Cantidad de Pruebas	Cantidad de clases	Promedio
Baja	48	61,53846154
Media	2	2,564102564
Alta	8	10,25641026

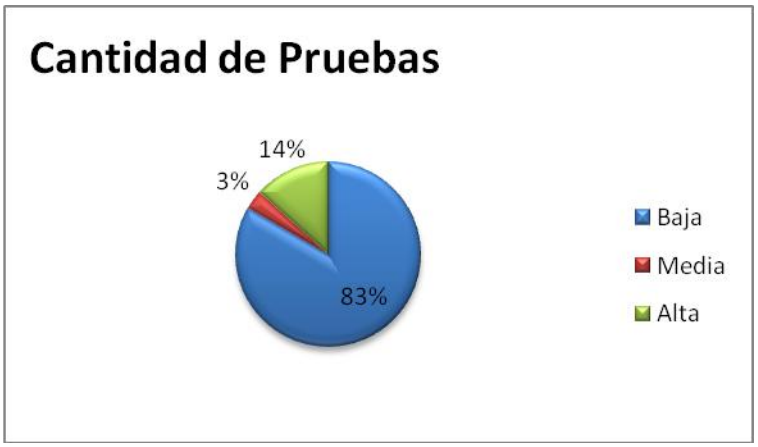


Figura 11 Representacion de la Cantidad de Pruebas

De manera general los resultados de esta métrica son positivos. El acoplamiento existente entre las clases es bajo, el 78% no tiene ningún acoplamiento, el 5% es bajo, el 3% es medio, y solo el 14% tiene alto acoplamiento. El nivel de reutilización de las clases es bastante bueno, el 83% de las clases pueden ser reutilizadas. La complejidad de mantenimiento y la cantidad de pruebas de las clases es baja.

4.3 Tamaño Operacional de Clases (TOC).

Tabla 31 Tamaño Operacional de Clases

Atributo que afecta	Modo en que lo afecta
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución en el grado de reutilización de la clase.

Luego de efectuar los cálculos para un:

Total de clases	58
Promedio de procedimientos	4,344827586

Se obtuvieron las siguientes gráficas:

Para la Responsabilidad.

Tabla 32 Responsabilidad de clases

Responsabilidad	Cantidad de clases	Promedio
Baja	48	92,30769231
Media	5	9,615384615
Alta	5	9,615384615

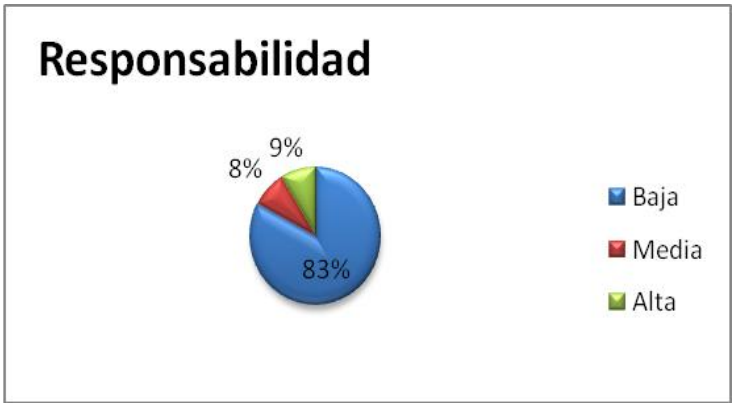


Figura 12 Representación de la Responsabilidad de Clases.

Para la Complejidad

Tabla 33 Complejidad de Clases.

Complejidad	Cantidad de clases	Promedio
Baja	48	92,30769231
Media	5	9,615384615
Alta	5	9,615384615

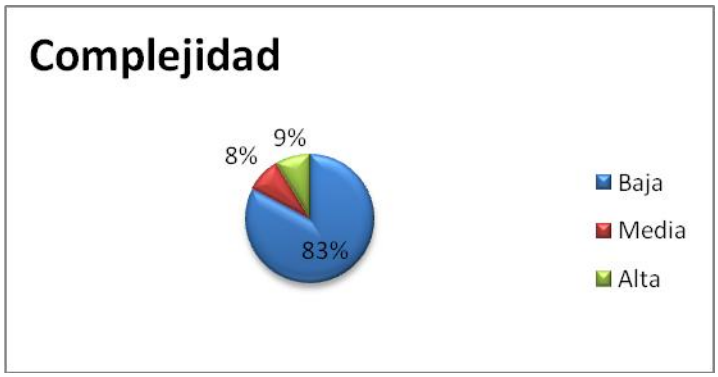


Figura 13 Representación de la Complejidad de Clases.

Para la Reutilización de Clases.

Tabla 34 Reutilización de Clases.

Reutilización	Cantidad de clases	Promedio
Alta	48	92,30769231
Media	5	9,615384615
Baja	5	9,615384615

Figura 14 Representación de la Reutilización de Clases.

De manera general los resultados de esta métrica son positivos. La responsabilidad existente entre las clases es baja, el 83% es bajo, el 8% es medio, y solo el 9% tiene responsabilidad alta. El nivel de reutilización de las clases es bastante bueno, el 83% de las clases pueden ser reutilizadas. La complejidad de mantenimiento de las clases es baja.

Cuando la responsabilidad de las clases entre media y baja es mayor que el 80% se puede decir que cumple con el patrón de alta cohesión, además de que presentan una reutilización alta.

Resumen de los resultados alcanzados en la aplicación de las métricas para validar el diseño.

El resultado de la aplicación de estas métricas demuestra que por lo general las clases no están cargadas en responsabilidad, existe bajo acoplamiento entre las mismas; así como un alto nivel de reutilización. Indican además que el diseño no es complejo, son bajas la complejidad de mantenimiento y la complejidad en las pruebas. Estos resultados confirman la calidad del diseño.

Conclusiones

En este capítulo se aplicaron métricas para la evaluación del diseño obtenido, dándole solución al objetivo planteado. El diseño propuesto no presenta una alta complejidad permitiendo que las pruebas no sean complejas y que el resto de los componentes puedan ser integrados sin muchas dificultades, además presenta un bajo acoplamiento. Se puede concluir que el diseño obtenido posee una calidad aceptable, facilitando la continuación eficiente del desarrollo en etapas posteriores.

Conclusiones generales

Con este diseño se le da cumplimiento al objetivo general trazado al inicio del desarrollo del trabajo, realizando el diseño del Motor de Procesamiento de Sentencias del Sistema Chronos que permita la réplica asíncrona multimaestro de sentencias y la recopilación de información para actividades de monitorización en un *cluster* de bases de datos Postgresql, además se cumplieron los objetivos específicos, existiendo una correspondencia entre los objetivos planteados y los resultados obtenidos:

- Se desarrolló el marco teórico de la investigación del diseño del Motor de Procesamiento de Sentencias del Sistema Chronos.
- Se identificaron los procesos relacionados con la réplica de datos en el sistema Chronos.
- Se diseñó el Motor de Procesamiento de Sentencias.
- Se validó el diseño realizado utilizando técnicas adecuadas para comprobar la calidad del mismo.

Recomendaciones

Este trabajo propone una solución al problema planteado inicialmente y satisface los objetivos trazados al inicio de la investigación, pero se recomienda añadir soporte para réplicas entre sistemas de Gestión de Bases de Datos diferentes de Postgresql como MySQL, Oracle y Microsoft SQL Server, añadir soporte para réplica entre diferentes plataformas como Windows. Se propone que este diseño del Motor de Procesamiento de Sentencias del sistema Chronos sea posteriormente implementado.

Bibliografía

1. CENTALAD. Centro de Tecnologías de Almacenamiento y Análisis de Datos. 2010. [Disponible en: http://portal.centalad.prod.uci.cu/]

2. Cholvi Juan, Vicente. Concurrencia y Sistemas Distribuidos. 2003. 348 pp.

3. M. Baker, R. Buyya, *Cluster* computing at a glance. Software Practice and Experience 29 (6), 1999, pp. 551-576.

4. Microsoft Corporation. Diccionario de Informática e Internet de Microsoft. (Español) Vuelapluma, S. L *trad.* ed. 2 ed. 2005, 880p.

5. C. J. Date. Introducción a los Sistemas de Bases de Datos. Pearson Prentice Hall, 2006, 480pp.

6. Aluziner. *Base de Datos Distribuidas.* 2010.[Disponible en: http://aluziner.googlepages.com/BD_Distribuidas.doc]

7. Lizama Mué, Yadira, Mario Michel Concepción Milanés. Disponibilidad y accesibilidad a datos en la plataforma Génesis. Universidad de las Ciencias Informáticas. Ciudad de la Habana. 2009. 93pp.

8. Ranade, Sandeep D. Asynchronous Replication. LINUX Journal, 2008. [Disponible en: http://www.linuxjournal.com/article/7265].

9. Douglas Dyllon Jeronimo de Macedo, Hilton Ganzo William Perantunes, Rafael Andrade, Aldo von Wangenheim, Mario Antonio Ribeiro Dantas, "Asynchronous Data Replication: A National Integration Strategy for Databases on Telemedicine Network," Computer-Based Medical Systems, IEEE Symposium on, 2008, pp. 638-643.

10. Sitio oficial de pgPool-II. *pgPool-II.* 2010.[Disponible en:http://pgpool.projects.postgresql.org/.]

11. Sitio oficial de PG*Cluster*. *PGCluster.* 2010.[Disponible en:http://pg*cluster*.projects.postgresql.org/.]

12. Sitio oficial de Slony-I. *Slony-I.* 2010.[Disponible en:http://www.slony.info/.]

13. Sitio oficial de Bucardo. *Bucardo.* 2010.[Disponible en:http://bucardo.org/wiki/Bucardo].

14. Luis Alberto Pimentel Gonzalez. *Reko.* 2010.[Disponible en: https://forge.uci.cu/gf/project/reko/].

15. Sitio oficial de postgreSql. *postgreSql.* 2010.[Disponible en: http://www.postgresql.org/docs/8.4/interactive/high-availability.html].

16. freedownloadmanager.*freedownloadmanager.* 2010. [Disponible en: http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/].

17. Sitio oficial de visual-paradigm. *visual-paradigm.* 2010.[Disponible en: http://www.visual-paradigm.com/product/vpuml/provides/ .]

18. Booch, G.: Rumbaugh, J. y Jacobson, I.; "El Lenguaje Unificado de Modelado". 2000. Addison-Wesley.320pp.

19. Pressman, Roger; Ingeniería de software. Un enfoque práctico. 2002. McGraw.Hill/Interamericana de España. Capítulos 1, 2,3 y 11. Páginas 3-48 y 181-195.

20. Jacobson, I.; Booch, G. y Rumbaugh, J.; "El Proceso Unificado de Desarrollo de software". 2000. Addison-Wesley. Capítulos 1-5. Páginas 3-104, 407-424.

21. Berard Laranjeira. *Capítulo 6. Métricas para Sistemas Orientados a Objetos.* 2010. [Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/gonzalez_d_h/capitulo6.pdf.]

22. Savannah. Libconfuse.2010. [Disponible en: http://savannah.nongnu.org/projects/confuse].

23. libpq-C Library. 2010. [Disponible en: http://www.PostgreSQL.org/docs/8.3/interactive/libpq.html].

24. Larman Craig. UML y Patrones, 1999 Tomo I Capítulos 18, Páginas 185-215.

25. RUMBAUGH JAMES. El lenguaje unificado de modelado. Manual de Referencia. Madrid, Pearson Educación. 2000. 526pp.

26. Fernández Aramayo, David Ricardo. Arquitectura de Software. Universidad Tecmilenio, ITESM.2000.560pp.

27. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, 1994, 383pp.

28. M. Townsend. Exploring the Singleton Design Pattern. 2010. Disponible en: http://msdn.microsoft.com/en-us/library/ms954629.aspx.

Anexos

Anexo 1. Estándares de Diseño de Base de Datos

Para la confección del modelo de datos del Sistema Chronos se siguieron los siguientes estándares de diseño de bases de datos:

1 Nombres de los objetos.

a) Los nombres de todos los objetos creados en la base de datos deben sugerir el significado de lo que representan en la vida real. Por ejemplo la tabla *tusuario* contiene la información de todos los *usuarios* del sistema. Se deben evitar términos ambiguos o que sugieran distintas interpretaciones.

b) Todos los nombres se escriben con letra minúscula y para nombres compuestos se separan las palabras con el guión bajo "_", por ejemplo *tnodo_servidor*, excepto para el caso de los identificadores que comienzan con "id" como el campo *idusuario* de la tabla *tusuario*. No deben abreviarse los nombres de los objetos.

c) Únicamente se utilizarán caracteres alfabéticos, salvo que por la naturaleza del nombre se necesiten dígitos numéricos. Se prohíbe el uso de caracteres de puntuación o símbolos.

d) Las letras acentuadas se reemplazarán con las equivalentes no acentuadas, y en lugar de la letra eñe (ñ) se utilizará (nn). Ejemplo: *descripcion* para la Descripción de un módulo de cálculo *y anno* para el Año de inscripción.

e) Deben agregarse comentarios a las bases de datos y los campos, sobre todo a los booleanos.

3. Campos de las tablas.

a) Ajustar al máximo el tamaño de los campos para no desperdiciar espacio en la base de datos

y seleccionar los tipos de datos apropiados.

b) Identificar las restricciones de campos no vacíos (NOT NULL), de llaves primarias y claves compuestas.

c) Los campos que representan la clave primaria deben ubicarse al inicio de las tablas para minimizar el tiempo en las operaciones de búsqueda.

d) El nombre del campo clave debe estar compuesto por “id” + nombre de la tabla. Dependiendo de la naturaleza de la entidad, el nombre de la tabla a usar es el de la misma tabla, o el de la relacionada. Ejemplos: tabla *tusuario* => *idusuario*.

e) Cuando en una tabla existen campos que tienen el mismo nombre, debe asociarse un alias que represente la información que almacenan. Ejemplo: En la tabla *tmensaje idusuario*=> *idusuario* que envía el mensaje e *idusuariodestino* => *idusuario* que recibe el mensaje.

Anexo 2. Estándares de código C++

Nomenclatura de las clases

1. El nombre de la clase debe ser un nombre claro, fonéticamente agradable.

2. Debe cumplir con las reglas de C++ para la nomenclatura.

3. En cuanto a estructura debe comenzar inicialmente con la letra "C" en mayúscula.

4. La letra "C" debe estar seguida una palabra con letra inicial mayúscula.

5. Si el nombre es compuesto se aplica lo anterior y se le concatena la otra palabra comenzando esta con letra inicial mayúscula.

6. No se debe utilizar el carácter "_" para los nombres de clases.

7. En el caso de las interfaces el nombre comenzará con "I" seguido de otra palabra con letra inicial mayúscula.

8. Nuca salvo excepciones bien justificadas el nombre de la clase aparecerá en plural.

9. Las llaves de la clase estarán alineadas a 0 espacios horizontales coincidiendo a verticalmente con la palabra reservada class. La segunda llave siempre irá seguida del carácter ";" que nunca debe faltar.

Nomenclatura de los métodos

1. Siempre se le declarará un método "Set" y un "Get" para cada atributo aunque se suponga a priori que no será utilizado.

2. El orden será primero todos los "Set" , luego todos los "Get" y luego las restantes funciones de la clase.

3. En caso de que existan funciones privadas estas se declararán debajo del último atributo declarado.

4. Debe ser fonéticamente agradable y expresar lo que hace de tan solo leerlo.

5. El nombre de un método siempre comenzará con una palabra con letra inicial mayúscula. En el caso de nombres compuestos se le concatenará la siguiente o siguientes palabras comenzando cada una con letra inicial mayúscula.

7. Si la función es para cambiarle el valor a un atributo se le pondrá delante el prefijo “Set” seguido del nombre del atributo. Entre paréntesis se le colocará el tipo de dato del atributo y el mismo nombre del atributo con el prefijo "p". Esta función será de tipo void.

8. Si la función es para retornar el valor a un atributo se le pondrá delante el prefijo “Get” seguido del nombre del atributo. Entre paréntesis no se le pasará ningún parámetro. Esta función será del tipo del atributo.

9. No se utilizará el carácter “_” para la nomenclatura de funciones, este queda reservado para otras nomenclaturas.

10. El nombre de los parámetros de los métodos debe cumplir con las reglas de formación de los atributos a lo cual se le añade que comenzarán con la letra inicial minúscula "p".

11. Los métodos se declararán con un espacio Tab a partir de 0 espacios con respecto a su modificador de visibilidad superior.

Glosario de Términos

Failback: En un sistema de red en *cluster* (uno con dos o mas servidores interconectados), es el proceso de restaurar los recursos y servicios a su servidor principal después de haber tenido que reubicarlos temporalmente en un sistema de emergencia mientras se llevan a cabo reparaciones en el host original.

Shell: Fragmento de software, normalmente un programa separado, que proporciona comunicación directa entre el usuario y el sistema operativo.

***Cluster*:** Conglomerado de computadoras conectadas por una red de alta velocidad.

Failover: Es un modo de operación de backup en el cual las funciones de un componente del sistema son asumidas por un segundo componente del sistema cuando el primero no se encuentra disponible debido a un fallo ó un tiempo de parada preestablecido. Es usado para hacer a los sistemas más tolerantes a fallos, y de esta forma hacer el sistema permanentemente disponible.

Middleware: Es un tipo de software o programa de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.

Nodo: En el área de la computación paralela representa una computadora en un *cluster*. Los *cluster* están compuestos por un conjunto de nodos.

Log: Mensaje emitido por el MPS que se almacena en un fichero físico.

Semáforo: Variable especial protegida que constituye el método clásico para restringir o permitir el acceso a recursos compartidos en un entorno de multiprocesamiento donde se ejecutarán varios procesos concurrentemente.

CRUD: Se refiere a los términos en inglés Create, Read, Update y Delete referidos a las operaciones de Crear, Recuperar, Actualizar y Eliminar respectivamente.