Universidad de las Ciencias Informáticas

Facultad 3

Título: Diseño e implementación de la herramienta de administración y configuración de Chronos.

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

Autor: Miguel Luis Sojo Hernández.

Tutor: Ing. Lissuan Fadraga Artiles.

Co-tutor: Ing. Sergio Hernández Cisneros.

La Habana, mes del 2012

Declaración de autoría

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

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

Autores:

_____

Miguel Luis Sojo Hernández.

Tutor:

_____

Ing. Lissuan Fadraga Artiles

Agradecimientos

A mis padres queridos:

A mi mamá por quererme tanto y darme día a día todo cuanto necesité para cursar mis estudios, siempre te estaré agradecido.

A mi papá por estar siempre ahí con el buen consejo y ser mi ejemplo a seguir en lo moral y profesional.

A toda mi familia:

A mis hermanos Made, Papa, Reinier, Angel, Omar y Tico, por alentarme a que terminara mis estudios.

A mis tías y tíos de La Habana y El Cobre: Maida, Marlene, Oris, Villita, Ana, Curro, Tati, Gladys y Valentín, por su aporte a que alcanzara esta meta.

A mis primas y primos: Niurka, Yanelis Yarita, Dayana, Dayanis, Taquechi, Odis, Flora, Néstor, Achito, Piquito, Yunior; a mis sobrinos (Adonis).

A mis amistades de Santiago Daril, Yasmany y Yeolkis.

En especial a todos mis compañeros de estudios, Daniel, Leyandry, Yudier, Cuadra y Jorge. Los profesores, Sergio, Eliober, Javier, Yadira y mi tutor Lissuan.

A todos muchas gracias.

Dedicatoria

Quisiera dedicar este trabajo de diploma a mis padres, por todo lo que han sacrificado para que pudiera graduarme como ingeniero en esta universidad, a mi familia y a esta maravillosa escuela obra de Fidel y la Revolución, que me ha formado como profesional y me ha permitido conocer personas de todo el país, junto a las que he pasado buenos momentos.

Resumen

En los actuales sistemas de software es necesario garantizar que toda la información que se gestiona esté disponible, y que ante fallos se recuperen de manera automática. En la Universidad de las Ciencias Informáticas (UCI) se han desarrollado sistemas que cumplen con estas características, muestra de ello es la creación de varios sistemas de réplica, entre los que se encuentra el Sistema Chronos, el cual cumple con las exigencias de sincronización y monitorización requeridas en nuestros días y se encuentra a la altura de otras herramientas de su tipo; pero dicho sistema cuenta con una aplicación de administración y configuración Web que dificulta la integración con la plataforma de digitalización DigiDAP perteneciente al centro CEGEL, debido a los requisitos de software que necesita para su funcionamiento. Como parte de la solución a esta problemática se realizó el presente trabajo de diploma cuyo objetivo es el diseño e implementación de una aplicación de escritorio para la administración y la configuración de Chronos, para ello se obtuvieron las historias de usuario, el plan de iteraciones y las tareas de ingeniería para su implementación. Esta herramienta brinda la posibilidad de administrar el Motor de Procesamiento de Sentencias (MPS), el cual es el núcleo del sistema de réplica Chronos, reconfigurar las sincronizaciones para añadirle cambios a las mismas y permite integrarse de manera más sencilla con la plataforma DigiDAP.

Palabras clave: réplica de datos, asincrónica, entorno multi-maestro, clúster.

Índice

Introducción

El auge vertiginoso de la informática y la necesidad de almacenamiento confiable de grandes volúmenes de información ha propiciado un aumento en la creación de herramientas de gestión de Base de Datos. Con el objetivo de contribuir a que estas herramientas logren una mejor seguridad de los datos que almacenan, disponibilidad de la información en el momento de ser requerida y mejora en el tiempo de respuesta, dado la gran cantidad de transacciones a las que son sometidos y debido a la centralización de la información en un único nodo, se ha desarrollado la técnica de réplica de datos. Esta técnica en nuestros días, viene aparejada a la creación de muchos de los procesos de negocio que se realizan, consiste en hacer una copia fiel de los datos de un nodo a otro o más nodos de respaldo y transmitirla de manera síncrona o asíncrona.

La réplica síncrona se realiza en sistemas que requieran una actualización constante de la información que se encuentra en una Base de Datos, asegurando la consistencia de los datos en todo momento, lo que provoca una sobrecarga en el sistema de redes debido al número elevado de procesos transaccionales que se realizan.

La réplica asíncrona contribuye al rápido funcionamiento de un sistema, debido a las limitaciones temporales de las operaciones de réplica, independientemente que no posibilite la coherencia de datos entre los distintos nodos.

La UCI se ha dado a la tarea de desarrollar diferentes sistemas, teniendo en cuenta los altos niveles de información que manejan los gestores en los distintos proyectos que se realizan. Entre estos sistemas de réplica se encuentran el sistema Reko, que representa una solución para la réplica de datos del Proyecto Sistema de Gestión Penitenciaria (SIGEP), y la solución Réplica Bidireccional basada en control de cambios, del Proyecto Identidad, caracterizados por darle soluciones de disponibilidad a los negocios para los que fueron creados.

Otro de los sistemas desarrollados es Chronos, compuesto por tres componentes: el MPS, la interfaz gráfica de usuario de administración y configuración y una librería dinámica que captura las transacciones que se realizan en el gestor de bases de datos. Dicha librería establece la comunicación con el MPS para las operaciones de réplica.

El Sistema Chronos es un software de réplica asíncrona para entornos multi-maestro, con características similares a las de otras herramientas de su tipo. Esta herramienta permite a todo sistema que utilice PostgreSql la administración de varias operaciones de transacción entre los nodos del clúster, a través de su interfaz de administración y configuración Web, replicando de forma asíncrona todas las sentencias SQL que se originan en un nodo a los otros, de manera tal que se garantice la disponibilidad de todos los datos en el momento que estos se requieran y la recuperación ante fallos. Sin embargo, la herramienta de administración y configuración web de Chronos necesita un servidor web como requisito indispensable para su funcionamiento que soporte el lenguaje PHP y las correspondientes librerías para la interacción con la base de datos y la ejecución de consultas, dificultando su adaptación a los distintos sistemas que requieran sus servicios; carece de una funcionalidad que permita la modificación de las sincronizaciones, o sea que si una cambia, no puede reconfigurarse, sino que hay que eliminarla y crearla nuevamente. Además, no permite la administración del MPS.

Debido a la **situación problemática** antes mencionada, surge la necesidad de darle solución al siguiente **problema:** ¿Cómo mejorar la administración y configuración de la herramienta de réplica Chronos?

El **objeto de estudio** que plantea esta investigación es: La gestión de Base de Datos. Enmarcado en el **campo de acción:** Los sistemas de administración de réplica de Base de Datos.

Se plantea como **objetivo general:** Desarrollar una herramienta que permita la administración y configuración de Chronos.

Para guiar la presente investigación se plantea la siguiente **idea a defender**: Si se desarrolla una herramienta de administración y configuración de réplica se mejorará la gestión de las transacciones y mantenimiento en el sistema Chronos.

Para darle cumplimiento al objetivo general se definieron los siguientes **objetivos específicos:**

1. Elaborar marco teórico de la investigación.
2. Obtener los elementos del diseño de la solución.
3. Realizar la implementación de los elementos de diseño obtenidos.
4. Validar la solución propuesta para garantizar la calidad de la misma.

Para responder a los objetivos específicos se realizarán las siguientes **tareas específicas:**

- ✓ Estudio del estado del arte de las herramientas de administración y configuración de Sistemas de Réplica existentes en la UCI.
- ✓ Análisis de los artefactos Especificación de Requisitos de Software e Historias de Usuario como entrada al diseño y la implementación a realizar.
- ✓ Especificación de las Tarjetas CRC y las tareas de ingeniería.
- ✓ Implementación de los elementos de diseño obtenidos.
- ✓ Validación de la solución propuesta a partir las pruebas de aceptación y unitarias.

Entre los **métodos teóricos** utilizados para la investigación se emplearon (1) [1]:

- ✓ Método **analítico-sintético***:* permite la división mental del fenómeno en sus múltiples relaciones y componentes para facilitar su estudio y establece la unión entre las partes previamente analizadas, posibilitando descubrir sus características generales y las relaciones esenciales entre ellas. Se utilizó en la realización del estudio teórico de la investigación y en la investigación previa sobre el funcionamiento del proceso de replicación de datos del sistema Chronos permitiendo extraer los elementos más importantes.
- ✓ Método **histórico-lógico:** Se utiliza para analizar la trayectoria completa de un objeto, su condicionamiento a los diferentes períodos de su historia, así como las etapas principales de su desenvolvimiento. Este método fue de vital importancia para la determinación del estado del arte de las herramientas, metodologías y tecnologías utilizadas.

El presente trabajo consta de 3 capítulos:

En el **Capítulo 1 Fundamentación Teórica**: se plantea la fundamentación teórica del tema, incluye un estudio del estado del arte abordando temas como las técnicas, metodologías y herramientas que se usan en la actualidad para la réplica en bases de datos PostgreSql a nivel nacional, haciendo énfasis en la Universidad de las Ciencias Informáticas como uno de los principales desarrolladores de software en Cuba.

En el **Capítulo 2 Diseño de la propuesta de Solución**: se describe la propuesta de solución del sistema,

identificando las historias de usuarios para darle cumplimiento a los requisitos funcionales y no funcionales del sistema, permitiendo estimar la prioridad y esfuerzo en cada una de ellas. Se realiza el Plan de iteraciones y entregas, y se define el diseño del sistema.

En el **Capítulo 3 Implementación y Prueba:** se aborda todo el proceso de implementación y pruebas de la solución propuesta, obteniéndose las tareas de programación o ingenierías por cada iteración y los casos de prueba correspondientes a cada historia de usuario y funcionalidades internas del sistema.

Como posibles resultados a alcanzar con el desarrollo de este trabajo están:

- ✓ Desarrollar una herramienta que permita la administración y configuración de Chronos.
- ✓ La documentación técnica asociada al desarrollo de la anterior herramienta.

Capítulo 1: Fundamentación teórica

1.1 Introducción

En este capítulo se definen los principales conceptos relacionados con la réplica de datos. Se exponen además, las principales características de algunas de las herramientas creadas en la UCI, sus rasgos distintivos, ventajas y desventajas; así como las tecnologías y metodologías utilizadas en la presente investigación.

1.2 Réplica de datos. Conceptos asociados.

La réplica de datos consiste en la transferencia de información creada o modificada en uno de los nodos del clúster de base de datos a los otros que lo integran, garantizando: una alta disponibilidad, tolerancia a fallos, rendimiento, reducción de la carga en la red y reducción de costo (2). Esta técnica comprende un número importante de conceptos que son necesarios para un mejor fundamento de esta investigación:

1.2.1 Sistemas Gestores de Bases de Datos (SGBD)

"Un SGBD es una colección de programas cuyo objetivo es servir de interfaz entre la base de datos, el usuario y las aplicaciones" (3). Las características que lo definen son la integridad, confidencialidad, seguridad y disponibilidad de los datos que almacenan, permitiendo un manejo claro, sencillo y ordenado de los mismos. Entre las funcionalidades que debe permitir se encuentran (4):

- ✓ Definir una base de datos: especificar tipos, estructuras y restricciones de datos.
- ✓ Construir la base de datos: guardar los datos en algún medio controlado por el mismo SGBD.
- ✓ Manipular la base de datos: realizar consultas, actualizarla, generar informes.

1.2.2 Clúster de base de datos

Es un grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio. Es un sistema de computadoras distribuido que permite el procesamiento de operaciones de réplica entre sus nodos a una

mayor velocidad (5).

De un sistema de este tipo se espera que presente combinaciones de los siguientes servicios:

- ✓ Alto rendimiento.
- ✓ Alta disponibilidad.
- ✓ Equilibrio de carga.
- ✓ Escalabilidad.

1.2.3 Tipos de réplicas

Los tipos de replicación varían según el momento en que se realiza una operación de escritura en alguno de los nodos del clúster; estos pueden ser de forma síncrona o asíncrona.

Replicación síncrona

La replicación síncrona, se utiliza frecuentemente cuando se desea que los datos se encuentren disponibles en todo instante, mediante disparadores, que se encargan de enviar la réplica de los datos en la misma transacción, obteniendo en la mayoría de los casos, un sistema sobrecargado debido a las constantes operaciones de escritura (6).

Replicación asíncrona

La replicación asíncrona, parte desde la peculiaridad de un sistema en el que no sea necesario una actualización constante de los datos, permitiendo que estos sean actualizados un instante posterior a su transacción, ya sea de manera instantánea o programada, tiene como desventaja que al momento de ser requerido determinado dato, este se encuentre caducado y provoque inconsistencias (6). Existen 2 tipos de réplica asíncrona:

- ✓ 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 (7).
- ✓ 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 clúster 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 (7).

1.2.4 Escenario de réplica

Un escenario de réplica no es más que el entorno en que se van a desarrollar las distintas operaciones de transacción de datos, y consiste en la selección del modo multi-maestro o maestro-esclavo según las necesidades objetivas del usuario o la institución que esté a cargo.

Multi-maestro

El entorno de replicación multi-maestro 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 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 multi-maestro es una alternativa eficaz para aquellos sistemas que realizan gran cantidad de operaciones de lectura/escritura sobre sus datos (7).

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).

1.3 Sistemas de Réplica para PostgreSql desarrolladas en Cuba y la UCI

Para la realización de este trabajo se propone el estudio de algunas de las diferentes herramientas que han sido creadas para los servidores y base de datos de PostgreSql en la UCI:

Reko: solución para la réplica de datos en el Proyecto Sistema de Gestión Penitenciaria (SIGEP).

Réplica bidireccional: basada en control de cambios del Proyecto Identidad.

Reko: soporta la replicación de transacciones a través de Hibernate y la replicación de acciones por 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 (8).

Réplica Bidireccional: 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 (9).

1.4 Tecnologías, metodologías y herramientas de software para el desarrollo del sistema

1.3.1 Metodologías de desarrollo de software

Una metodología de desarrollo de software es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información, a través de herramientas, modelos y métodos. Conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software (9).

Proceso Unificado de Rational (RUP)

Proceso Unificado de Desarrollo, por sus sigla en inglés (RUP), es una metodología de desarrollo que tiene como finalidad, la mejora de las prácticas que se implementan en el desarrollo de software, para obtener un sistema de calidad, que cumpla con los requerimientos de los usuarios, entre las características que posee están: cubre todo el ciclo de vida de desarrollo del software, utiliza UML, como lenguaje de modelado visual, desarrollo iterativo, administración de los requerimientos, uso de arquitectura basada en componentes, verificación continua de la calidad, y administración del cambio, está organizado en 4 fases (Inicio, Elaboración, Construcción, Transición) secuenciales y por nueve disciplinas, que independientemente que no se realicen con la misma intensidad en cada una de las fases, están presentes en todo el desarrollo del software (10).

Programación Extrema o XP

Programación Extrema, por sus siglas en inglés (XP), es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico, presenta ciclos cortos de trabajo y una generación de artefactos reducida (11).

Producto del análisis de ambas metodologías, se escoge XP cómo guía para el desarrollo del sistema a implementar, debido a la generación reducida de artefactos, lo que contribuye al corto tiempo que existe para diseñar e implementar el sistema. Además, el equipo de desarrollo contará con el asesoramiento constante del cliente.

1.3.2 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 (12).

Se determinó utilizarlo en su versión 2.0, producto de su capacidad de visualizar, especificar, construir y documentar los artefactos de un sistema, además de modelar sus funciones e integrarse fácilmente con la metodología y herramienta CASE seleccionadas [2].

1.3.3 Herramientas Case (Ingeniería de software asistida por computadoras)

Las herramientas Case son un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo del sistema de información, completamente o en algunas fases. Entre los beneficios que aportas este tipo de herramientas, se pueden observar (13):

- ✓ Potencia la mejora del producto final.
- ✓ Facilita el desarrollo de los procesos.
- ✓ Mejora la calidad del sistema.
- ✓ Disminución de tiempo.
- ✓ Garantizar la consistencia de los procedimientos.
- ✓ Captura de los datos del sistema.

Visual Paradigm

Se eligió Visual Paradigm en su versión 3.4, por ser una Herramienta CASE que provee el modelado de procesos de negocios a través del lenguaje UML, además de un generador de mapeo objetos-relacionales para los lenguajes de programación Java .NET y PHP. Es multiplataforma, muy potente y fácil de utilizar. Puede generar código automático, bases de datos y generar reportes, vinculándose estrechamente con el servidor de base de datos PostgreSql y el entorno de desarrollo NetBeans (escogidos para el desarrollo). Permite la especificación y modelación de las Tarjetas CRC (14)[3].

1.3.4 Sistemas de gestión de BD (PostgreSql)

PostgreSql es un poderoso sistema de gestión de bases de datos objeto-relacional de código abierto y debido a esta característica es un sistema orientado a objetos por lo que utilizan la herencia, tipo de datos,

funciones, restricciones, disparadores, reglas e integridad transaccional. Usa una arquitectura orientada a objetos para manejar procesos desde un cliente externo. Es compatible con todos los Sistemas Operativos, Incluyendo Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), y Windows. Fue seleccionado en su versión 8.4 debido a las características planteadas, su integración con el Visual Paradigm y NetBeans, y que el sistema Chronos es una solución de réplica para dicho gestor [4].

1.3.5 Tecnologías de Java

Java es un lenguaje de programación compilado e interpretado, donde todo programa debe compilarse y el código que se genera (bytecodes) es interpretado por una máquina virtual. De este modo se consigue la independencia de la máquina, el código compilado se ejecuta en máquinas virtuales que si son dependientes de la plataforma, aunque existe una máquina virtual (intérprete) para cada una de estas plataformas. Es un lenguaje orientado a objetos de propósito general y se puede utilizar para construir cualquier tipo de proyecto (15), se puede acceder a bases de datos fácilmente con JDBC independientemente de la plataforma utilizada. El manejo de las bases de datos es uniforme, es decir transparente y simple.

Máquina Virtual Java

Una Máquina Virtual Java (en inglés Java Virtual Machine, JVM) es un programa nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un Código binario especial (bytecode), el cual es generado por el Compilador del lenguaje Java.

La JVM es una de las piezas fundamentales de la plataforma Java. Básicamente se sitúa en un nivel superior al Hardware del sistema sobre el que se pretende ejecutar la aplicación, y este actúa como un *puente* que entiende tanto el bytecode, como el sistema sobre el que se pretende ejecutar. Así, cuando se escribe una aplicación Java, se hace pensando que será ejecutada en una máquina virtual Java en concreto, siendo ésta la que en última instancia convierte de código bytecode a código nativo del dispositivo final. La gran ventaja de la máquina virtual java es aportar portabilidad al lenguaje de manera que desde Sun Microsystems se han creado diferentes máquinas virtuales java para diferentes arquitecturas y así un programa .class escrito en un Windows puede ser interpretado en un entorno Linux. Tan sólo es necesario disponer de dicha máquina virtual para dichos entornos. De ahí el famoso axioma

que sigue a Java, "escríbelo una vez, ejecútalo en cualquier parte", o "Write once, run anywhere" (16).

JRE

El JRE (Java Runtime Environment, o Entorno en Tiempo de Ejecución de Java) es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java. El usuario final usa el JRE como parte de paquetes software o plugins (o conectores) en un navegador Web. Sun ofrece también el SDK de Java 2, o JDK (Java Development Kit) en cuyo seno reside el JRE, e incluye herramientas como el compilador de Java, Javadoc para generar documentación o el Depurador. Puede también obtenerse como un paquete independiente, y puede considerarse como el entorno necesario para ejecutar una aplicación Java, mientras que un desarrollador debe además contar con otras facilidades que ofrece el JDK (16).

JDK

JDK (Java Development Kit o Herramientas de Desarrollo en Java), es un software que provee herramientas de desarrollo para la creación de programas en Java. Puede instalarse en una computadora local o en una unidad de red. Se puede definir como un conjunto de herramientas, utilidades, documentación y ejemplos para desarrollar aplicaciones Java (16). Para el desarrollo del sistema se seleccionó la versión 1.0.6_26.

1.3.6 Entornos de Desarrollo

NetBeans IDE

Se determinó escoger NetBeans en su versión 7.0.1, es un Entorno de Desarrollo Integrado (IDE) y código abierto. Escrito en el lenguaje de programación Java, soporta el desarrollo de todos los tipos de aplicación en el lenguaje elegido (J2SE, J2EE, Web, EJB y aplicaciones móviles), se integra fácilmente con las herramientas seleccionadas Visual Paradigm y PostgreSql para la generación de entidades persistentes, a través de la interfaz de programación de aplicaciones (API) JPA, que proporciona un modelo de persistencia basado en POJO (Plain Old Java Object) para mapear bases de datos relacionales en Java [5] y permite mejorar la apariencia de los componentes diseñados utilizando Swing, que es una biblioteca gráfica que forma parte de la Fundación de Clases de Java (JFC por sus siglas en inglés) y provee facilidades para ayudar a construir GUI´s (Interfaz Gráfica de Usuario o Graphical User Interface en inglés)

[6].

1.3.7 JUnit

JUnit es una herramienta simple de código abierto, que se ha convertido en el estándar para probar clases java de forma unitaria. Es fácil de usar y configurar por lo que prácticamente no existe curva de aprendizaje. JUnit está diseñado para reportar fallos o éxitos sobre un código sin necesidad de interpretar el reporte. Se seleccionó esta herramienta en su versión 4.5 por su fácil integración con el lenguaje Java y NetBeans [7].

1.5 Conclusiones parciales

Con la realización de este capítulo se pudieron definir los distintos conceptos asociados a la réplica de datos, así como el estado del arte de las distintas herramientas, metodologías y tecnologías para el desarrollo de software en la actualidad, de los cuales es importante aclarar que en su mayoría, los motivos por los que se escogieron se adaptan a los de la plataforma de digitalización del centro CEGEL con la cuál quedaría integrado el Sistema Chronos. Se realizó además el estudio de algunas de las herramientas existentes en la actualidad, similares a la que se pretende realizar.

Se definieron utilizar las siguientes herramientas, tecnologías y metodología:

- ✓ como metodología de desarrollo se seleccionó XP.
- ✓ se eligió UML v2.0 como lenguaje de modelado.
- ✓ como herramienta CASE, Visual Paradigm en su versión 3.4.
- ✓ el gestor de bases de datos que se escogió es PostgreSql v8.4.
- ✓ lenguaje de programación Java en su v1.6.
- ✓ el IDE de desarrollo NetBeans v7.0.1.
- ✓ y como herramienta para la realización de pruebas unitarias JUnit v4.5.

Capítulo 2: Diseño de la propuesta de solución

2.1 Introducción

La plataforma DigiDAP presenta la necesidad de integrarse con la herramienta de réplica Chronos, ya que esta necesita la técnica de réplica de la información debido a la cantidad de datos que almacena y el tiempo de respuesta ante la gran cantidad de consultas a las que serán sometidos, se realiza un cambio de tecnología en la interfaz de administración, transformándola en una interfaz de escritorio que mejorará la administración y configuración de la herramienta Chronos; se pretende además la mejora de algunas de las funciones que existen en esta herramienta, así como la adición de otras que le permitan un mejor funcionamiento. En este capítulo se expone brevemente una descripción general de la propuesta de sistema, como deben funcionar cada uno de los procesos del negocio que lo integran y la especificación de los requisitos funcionales y no funcionales del software en cuestión.

2.2 Propuesta del sistema

El sistema de réplica de bases de datos Chronos es una herramienta de réplica asíncrona para entornos multi-maestro. Chronos ha sido diseñado a partir de necesidades particulares de un escenario de réplica (replicación asíncrona para un entorno multi-maestro), asumiendo las principales potencialidades de las soluciones similares en la actualidad para PostgreSql e implementando nuevas mejoras que la distinguen del resto. Entre los componentes que la integran se encuentran:

LibTransaccionesSQL.so: Librería de enlace dinámico que extiende la funcionalidad del gestor capturando los cambios que se producen en cualquiera de las tablas contenidas.

Motor de Procesamiento de Sentencias (MPS): Constituye el núcleo de la aplicación, deberá replicar los cambios ocurridos en los Sistemas Gestores de Bases de Datos (SGBD) PostgreSql configurados por la herramienta de administración de Chronos y utilizará la librería Libtransaccionesql.so para capturar los cambios que se producen en cualquiera de las tablas de la Base de Datos.

Herramienta de Administración y Configuración: Es la aplicación de administración y configuración que se utiliza para la monitorización en tiempo real de Chronos. Es una aplicación Web a la que solo pueden

acceder y tener permiso de ejecución los usuarios previamente autenticados que tengan el rol de administradores de la misma, esta herramienta no cuenta con un módulo de administración y configuración del MPS que le permita activar, reiniciar, y apagar los procesos de réplica del mismo, y dificulta la integración con la plataforma DigiDAP.

Este trabajo propone el desarrollo de un sistema de escritorio en Java que permita la administración y configuración de la herramienta de réplica Chronos, gestionar las distintas Bases de Datos que se agreguen, las fuentes, grupos, sincronizaciones y distintos errores que puedan ocasionarse a través de un gestor de mensajes; además de facilitar la integración con DigiDAP y su posterior utilización por otros sistemas que requieran sus servicios. El nuevo sistema de escritorio permitirá realizar las sincronizaciones manteniendo el tipo y entorno de réplica utilizado en la herramienta anterior (replicación asíncrona para un entorno multi-maestro), así como la administración del motor de procesamiento y las reconfiguraciones de las sincronizaciones que se realicen.

2.3 Especificación de los requisitos

2.3.1 Requisitos funcionales

Los requisitos funcionales son las condiciones o capacidades que el sistema debe cumplir. Los requisitos funcionales que debe cumplir la aplicación son los siguientes:

- ✓ RF1 Adicionar Base de datos (BD).
- ✓ RF2 Mostrar BD.
- ✓ RF3 Modificar BD.
- ✓ RF4 Eliminar BD.
- ✓ RF5 Adicionar Grupo.
- ✓ RF6 Mostrar Grupos.
- ✓ RF7 Modificar Grupo.
- ✓ RF8 Eliminar Grupo.
- ✓ RF9 Adicionar Fuentes.
- ✓ RF10 Mostrar Fuentes.
- ✓ RF11 Eliminar Fuentes.
- ✓ RF12 Adicionar Sincronización.

- ✓ RF13 Mostrar Sincronizaciones.
- ✓ RF14 Reconfigurar Sincronización.
- ✓ RF15 Eliminar Sincronización.
- ✓ RF16 Eliminar Mensajes.
- ✓ RF17 Mostrar Mensajes.
- ✓ RF18 Buscar Mensajes.
- ✓ RF19 Administrar Motor de Procesamiento de Sentencias (MPS).
- ✓ RF20 Añadir Filtros de Réplica.
- ✓ RF21 Autenticar Usuario.

2.3.2 Requisitos no funcionales

Apariencia o interfaz externa

✓ El sistema debe contar con una interfaz amigable y sencilla, enfocada a los requerimientos finales del usuario, deberá permitir monitorizar todas las sincronizaciones y el estado de las réplicas de manera sencilla y rápida.

Usabilidad

✓ Debido al ambiente sencillo y amigable que se requiere, el usuario podrá navegar a través del sistema con conocimientos básicos en computación.

Fiabilidad

✓ El sistema estará disponible 24 horas al día, 7 días a la semana.

Rendimiento

✓ La velocidad de procesamiento de la información debe ser rápida, acorde con las necesidades del usuario.

✓ Aplicación de las diferentes técnicas de elaboración en el sistema para facilitar el rápido acceso a los datos.

Soporte

✓ El software deberá permitir posteriores modificaciones y actualizaciones por el

administrador y se debe garantizar el mantenimiento de la aplicación.

Seguridad

- ✓ La información manejada por el sistema debe estar protegida de acceso no autorizado y divulgación.
- ✓ La información manejada por el sistema será objeto de cuidadosa protección contra la corrupción y estados inconsistentes.
- ✓ A los usuarios autorizados se les garantizará el acceso al sistema y que los dispositivos o mecanismos utilizados para lograr la seguridad no ocultarán o retrasarán a los mismos para realizar las transferencias necesarias en un momento dado.
- ✓ Se deben crear usuarios que tendrán asignados permisos de acción sobre cada información manejada por el sistema para lo cual se requiere la autenticación del usuario.

Requerimientos mínimos de hardware

Estación de trabajo cliente:

- ✓ 1 GB de RAM, DDR2 667 MHz.
- ✓ Procesador de Pentium 4.
- ✓ 20 GB de disco duro, SATA 5400 RPM.
- ✓ Adaptador de red Ethernet 100 Mbps.
- ✓ Sistema de Energía Ininterrumpida (UPS) 500 Va.

Requerimientos de software

- ✓ El servidor de BD PostgreSql 8.4 tiene que estar instalado en los nodos donde se va a configurar la réplica.
- ✓ La librería de captura de cambios libtransaccionesql.so debe encontrarse en el directorio de librerías de PostgreSql.
- ✓ En la máquina donde se aloja la herramienta de administración y configuración debe tener la máquina virtual de java JRE.

Requerimientos en el diseño y la implementación

Para organizar el diseño y la implementación del sistema se utilizará:

- ✓ La metodología XP.
- ✓ UML como lenguaje de modelado.
- ✓ Visual Paradigm como herramienta CASE.
- ✓ Java será el lenguaje de programación a ser usado para la implementación.
- ✓ NetBeans como IDE de desarrollo.
- ✓ Los nombres de los métodos y de las clases deberán usar el estilo de codificación camello para lograr un mejor entendimiento entre el equipo de desarrolladores.

2.4 Fase de exploración o planificación

En esta fase, los clientes plantean a grandes rasgos las historias de usuarios (HU) que son de interés para la primera entrega del producto. Al mismo tiempo, el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto (17).

En esta fase, el cliente definió lo que necesita mediante la redacción de sencillas historias de usuarios, se clasificaron las mismas atendiendo a su prioridad en el negocio, se estimó su duración y confeccionó el Plan de iteraciones. Los programadores estimaron el tiempo de desarrollo de cada historia de usuario sobre la base de la información obtenida del cliente.

2.4.1 Historias de Usuario

Las historias de usuario son utilizadas por la metodología XP como una técnica para especificar los requisitos de la misma, tanto requisitos no funcionales como funcionales. Se trata de tarjetas en las cuales el usuario describe las características que el sistema debe poseer, las mismas pueden agrupar uno o varios requisitos funcionales identificados. XP recomienda que las HU sean escritas por el propio cliente. Es por ello que debe existir una buena relación entre los clientes y el grupo de desarrollo, la falta de comunicación entre ambos factores influye directamente en el fracaso del proyecto. La duración de una HU está dada por la suma del tiempo de cada una de las tareas en las que se divide una historia, su nivel

de prioridad debe ser especificado por el cliente, al igual que debe participar en cada una de las iteraciones de modo que se garantice una implementación que cumpla con la satisfacción de sus necesidades (18).

Clasificación de la prioridad de las HU.

Alta: Se le otorga esta prioridad a las HU que el cliente define como funcionalidades principales en el desarrollo del sistema y fundamentales para control general del sistema.

Media: Se le otorga a las HU que el cliente define como funcionalidades menos significativas, sin que estas afecten el desarrollo del sistema.

Baja: Se le otorga a las HU que se definen como funcionalidades de ayuda a la estructura y control de elementos asociados al desarrollo del sistema.

2.4.2 Definición de Historias de Usuario

Historias de usuario	Nombre	Prioridad
HU-1	Adicionar Base de Datos	Alta
HU-2	Mostrar Bases de Datos	Alta
HU-3	Modificar Base de Datos	Alta
HU-4	Eliminar Base de Datos	Alta
HU-5	Adicionar Grupo de réplica	Alta
HU-6	Mostrar Grupos de réplica	Alta
HU-7	Modificar Grupo de réplica	Alta
HU-8	Eliminar Grupo de réplica	Alta
HU-9	Adicionar Fuentes de réplica	Alta

HU-10	Mostrar Fuentes de réplica	Alta
HU-11	Eliminar Fuentes de réplica	Alta
HU-12	Adicionar Sincronización	Alta
HU-13	Mostrar Sincronizaciones	Alta
HU-14	Reconfigurar Sincronización	Alta
HU-15	Eliminar Sincronización	Alta
HU-16	Eliminar Mensajes	Media
HU-17	Mostrar Mensajes	Media
HU-18	Buscar Mensajes	Media
HU-19	Administrar Motor de procesamiento de Sentencias (MPS)	Baja
HU-20	Añadir Filtros de réplica	Alta
HU-21	Autenticar Usuario	Alta

Tabla 1. Definición de historias de usuario

2.4.3 Descripción de Historias de Usuario

Durante el análisis en la fase de exploración fueron identificadas 21 HU, cada una de ellas corresponden a las diferentes funcionalidades con las que contará la aplicación. A continuación se describen 3 de las HU más significativas:

No.:1	**Nombre:** HU-21 Autenticar Usuario

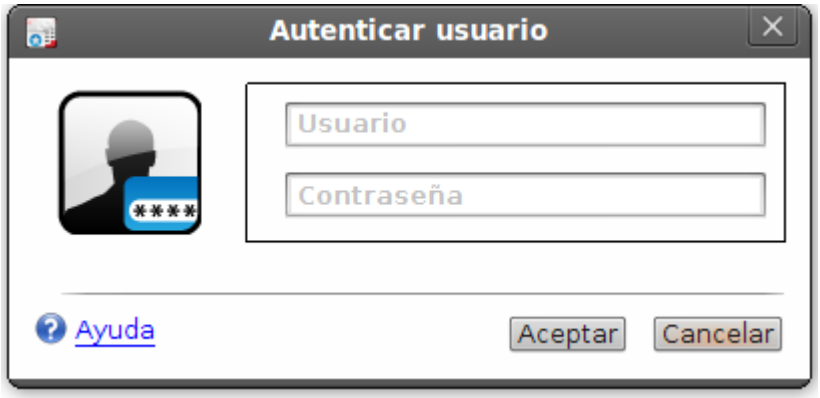
Usuario: Administrador	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Alto
Estimación: 2 día (s)	**Iteración asignada: 1**
Descripción: El usuario introduce las credenciales.	
Información adicional (observaciones):	
Prototipo de interfaz: 	

Tabla 2. Descripción de historia de usuario: Autenticar Usuario

No.:2	**Nombre:** HU-12 Adicionar Sincronización
Usuario: Administrador	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Alto
Estimación: 3 día (s)	**Iteración asignada:** 2
Descripción: El usuario se registra, selecciona la opción “Adicionar sincronizaciones”, y verifica los datos necesarios para añadir una nueva sincronización en el sistema.	

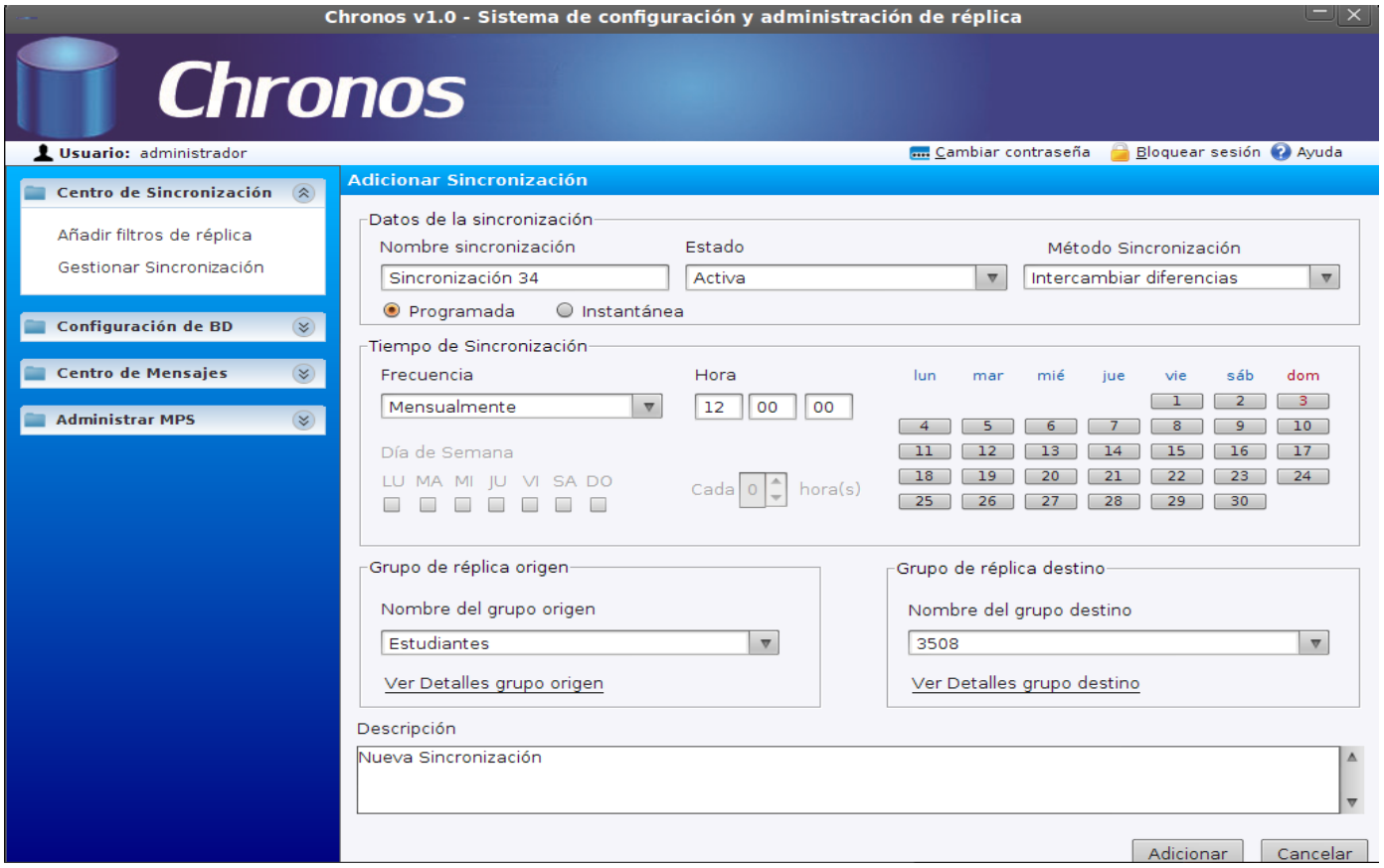
Información adicional (observaciones):
Prototipo de interfaz 

Tabla 3. Descripción de historia de usuario: Adicionar Sincronización

No.:3	**Nombre:** HU-1 Adicionar base de datos
Usuario: Administrador	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Medio
Estimación: 3 días	**Iteración asignada: 1**

Descripción: El usuario se registra, selecciona la opción “Adicionar base de datos” verifica los datos necesarios para añadir una nueva base de datos en el sistema, comprobando si tiene conexión primero con la misma.
Información adicional (observaciones):
Prototipo de interfaz 

Tabla 4. Descripción de historias de usuario: Adicionar base de datos

Para consultar las restantes HU, ver **Anexo 1**.

2.4.4 Estimación de esfuerzo por Historias de usuarios

Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos, aunque se definió para este trabajo estimar los puntos en días por la corta duración de las historias.

Historias de Usuario	Estimación (días)
Adicionar Base de Datos	3
Mostrar Bases de Datos	1
Modificar Base de Datos	2

Eliminar Base de Datos	1
Adicionar Grupo de réplica	1
Mostrar Grupos de réplica	1
Modificar Grupo de réplica	1
Eliminar Grupo de réplica	1
Adicionar Fuentes de réplica	3
Mostrar Fuentes de réplica	1
Eliminar Fuentes de réplica	3
Adicionar Sincronización	3
Mostrar Sincronizaciones	1
Reconfigurar Sincronización	3
Eliminar Sincronización	2
Eliminar Mensajes	1
Mostrar Mensajes	1
Buscar Mensajes	1
Administrar Motor de procesamiento de Sentencias (MPS)	3
Añadir Filtros de réplica	3

Autenticar Usuario	2

Tabla 5. Estimación de esfuerzo

2.4.5 Plan de iteraciones

Un Plan de Iteraciones consiste en organizar las historias de usuarios en función de las necesidades del cliente y del riesgo asociado a cada una de ellas, priorizando las mismas según la cantidad de iteraciones que se realicen. El cliente decidirá qué tareas realizar en cada iteración y en qué orden hacerlo, mientras el equipo de programadores lo asesorará, estimando el grado de complejidad y el tiempo que va a llevar cada parte del proyecto y solo el cliente aclara dudas que surjan sobre las especificaciones en el momento en que se determine como realizarlas, así como los recursos a emplear en cada una para darle solución (19).

- ✓ **Iteración 1**

Esta iteración tiene como objetivo la implementación de algunas de las HU de prioridad alta, debido a que las HU por la que está compuesto este sistema pertenecen a este tipo en su mayoría; permitiendo que una vez terminada esta iteración se tenga un prototipo con las funcionalidades principales para el funcionamiento del sistema.

- ✓ **Iteración 2**

Esta segunda iteración propone resolver las HU de prioridad alta que debido a su complejidad fue necesario no incluirlas en la primera iteración, para dividir el esfuerzo y el riesgo de implementación de cada una de ellas.

- ✓ **Iteración 3**

En esta iteración se implementarán las HU de prioridad baja y media, las mismas tienen como propósito brindarle comodidad al cliente en la gestión de otras tareas asociadas a las de prioridad alta. Al terminar esta iteración se obtendrá una versión 1.0 del producto final, listo para poner en funcionamiento y ser evaluado.

2.4.6 Plan de duración de las iteraciones

El plan de duración de las iteraciones se encarga de detallar el orden en que se implementarán las HU dentro de cada iteración y la estimación en semanas de la misma.

Iteraciones	Orden de las HU a implementar	Duración de las iteraciones
Iteración 1	Autenticar Usuario Adicionar Base de Datos Mostrar Bases de Datos Modificar Base de Datos Eliminar Base de Datos Adicionar Grupo de réplica Mostrar Grupos de réplica Modificar Grupo de réplica Eliminar Grupo de réplica Adicionar Fuentes de réplica Mostrar Fuentes de réplica Eliminar Fuentes de réplica	20 días
Iteración 2	Adicionar Sincronización Mostrar Sincronizaciones Reconfigurar Sincronización Eliminar Sincronización Añadir Filtros de réplica	12 días

Iteración 3	Eliminar Mensajes Mostrar Mensajes Buscar Mensajes Administrar Motor de procesamiento de Sentencias (MPS)	6 días
Total		**38 días**

Tabla 6. Plan de duración de las iteraciones

2.4.7 Plan de entregas

El plan de entregas establece qué HU serán agrupadas para conformar una entrega, y el orden de las mismas, así como la fecha fin de cada iteración. Este plan acopla las funcionalidades referentes a un mismo tema en el módulo que serán implementadas (20).

Historia de Usuario	**Fin de la 1ra iteración** **4ta semana de abril**	**Fin de la 2da iteración** **2da semana de mayo**	**Fin de la 3ra iteración** **4ta semana de mayo**	**Entrega**
Autenticar Usuario	V 0.1	Finalizado	Finalizado	**ENTREGA 1**
Gestionar Bases de Datos**(Adicionar, Mostrar, Modificar y Eliminar)**	V 0.1	Finalizado	Finalizado	
Gestionar Grupo de réplica**(Adicionar, Mostrar, Modificar y Eliminar)**	V 0.1	Finalizado	Finalizado	

Gestionar Fuentes de réplica**(Adicionar, Mostrar, Modificar y Eliminar)**	V 0.1	Finalizado	Finalizado	
Gestionar Sincronización **(Adicionar, Mostrar, Reconfigurar y Eliminar)**	-	V 0.2	Finalizado	ENTREGA 2
Añadir Filtros de réplica	-	V 0.2	Finalizado	
Gestionar Mensajes **(Mostrar, Buscar y Eliminar)**	-	-	V 1.0	ENTREGA 3
Administrar Motor de procesamiento de Sentencias (MPS)	-	-	V 1.0	

Tabla 7. Plan de entregas

2.5 Diseño del sistema

La Metodología XP se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo. Todo esto propició que el diseño de este sistema no requiera de la utilización de extensos diagramas de modelado. En su lugar se establecieron las tarjetas CRC (Contenido, Responsabilidad y Colaboración), las cuales son las encargadas de establecer la relación entre los objetos y cada una de las clases que componen el sistema (21).

2.5.1 Tarjetas CRC

El principal objetivo que presentan las tarjetas CRC, es permitir que todo el equipo participe en el diseño

de la aplicación. Las tarjetas CRC permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos.

Para poder diseñar el sistema utilizando las tarjetas, se debe cumplir con tres principios: Cargo o Clase, Responsabilidad y Colaboración (CRC). Están constituidas por 5 secciones en las que se mostrarán el nombre de la clase, el nombre de la clase padre en la herencia, el nombre de la(s) clase(s) hija(s) en la herencia, las responsabilidades o funcionalidades que esta clase debe permitir y las clases que le permiten realizar sus funcionalidades (colaboraciones) (21).

Debido a la facilidad de uso y entendimiento que establecen dichas tarjetas, se decidió utilizarlas para diseñar la aplicación que se desea desarrollar.

Tarjeta CRC: Nombre de la clase	
Clase: Nombre de la clase que se está modelando	
Súper clase: Nombre de la clase padre en la herencia	
Sub clase(s): Nombre de la(s) clase(s) hija en la herencia	
Responsabilidades: Es una descripción de alto nivel del propósito de la clase	**Colaboraciones:** Indica con cuales otras clases se requiere relación para cumplir la responsabilidad

Tabla 8. Plantilla de tarjetas CRC

Tarjeta CRC: AccionGestionarBD
Clase: AccionGestionarBD
Súper clase: AccionPanel
Sub clase(s): -

Responsabilidades:	**Colaboraciones:**
inicializarFormulario()	GestorNCBD
btnBuscar()	AdminChronosi18n
modificarBD()	BD
adicionarBD()	AccionModificarBD
eliminarBD()	AccionAdicionarBD
getFormulario()	GestorMarcoTrabajo
internacionalizacion()	GestorMensajes

Tabla 9. Tarjeta CRC: AccionGestionarBD

Tarjeta CRC: AccionGestionarGrupos	
Clase: AccionGestionarGrupos	
Súper clase: AccionPanel	
Sub clase(s): -	
Responsabilidades:	**Colaboraciones:**
inicializarFormulario()	GestorNCGrupo
btnBuscar()	AdminChronosi18n
onBtnSiguiente()	Grupo
eliminarGrupo()	AccionMostrar_SeleccionarBD
modificarGrupo()	AccionModificarGrupo
adicionarGrupo()	AccionAdicionarGrupo
getFormulario()	GestorMarcoTrabajo
internacionalizacion()	GestorMensajes

Tabla 10. Tarjetas CRC: AccionGestionarGrupos

Tarjeta CRC: AccionGestionarMensajes
Clase: AccionGestionarMensajes

Súper clase: AccionPanel	
Sub clase(s): -	
Responsabilidades: inicializarFormulario() btnBuscar() eliminarMensaje() reporte() verDetalles() getFormulario() internacionalizacion()	**Colaboraciones:** GestorNCMensajes AdminChronosi18n Mensaje TipoMensaje dsMensajes AccionDialogoVisorReporte AccionVerDetalles GestorMarcoTrabajo GestorMensajes

Tabla 11. Tarjetas CRC: AccionGestionarMensajes

Tarjeta CRC: AccionAdicionarFuentes	
Clase: AccionAdicionarFuentes	
Súper clase: AccionPanel	
Sub clase(s): -	
Responsabilidades: inicializarFormulario() onBtnBuscar() onBtnAnterior() onBtnTerminar() adicionarFuenteGrupo() adicionarFuentes() getFormulario() internacionalizacion()	**Colaboraciones:** GestorNCFuente AdminChronosi18n Fuente BD Grupo GestorMarcoTrabajo AccionVerDetalles GestorMensajes

Tabla 12. Tarjetas CRC: AccionAdicionarFuentes

Tarjeta CRC: AccionAdicionarBD	
Clase: AccionAdicionarBD	
Súper clase: AccionDialogo	
Sub clase(s): -	
Responsabilidades: inicializarFormulario() onBtnAceptar() onBtnCancelar() combrobarBD() getFormulario() internacionalizacion()	**Colaboraciones:** GestorNCBD BD AdminChronosi18n GestorMarcoTrabajo AccionGestionarBD GestorMensajes

Tabla 13. Tarjetas CRC: AccionAdicionarBD

Tarjeta CRC: AccionModificarBD	
Clase: AccionModificarBD	
Súper clase: AccionDialogo	
Sub clase(s): -	
Responsabilidades: inicializarFormulario() onBtnAceptar() onBtnCancelar() modificarBD() getFormulario() internacionalizacion()	**Colaboraciones:** GestorNCBD BD AdminChronosi18n GestorMarcoTrabajo AccionGestionarBD GestorMensajes

Tabla 14. Tarjetas CRC: AccionModificarBD

Tarjeta CRC: GestorNCBD	
Clase: GestorNCBD	
Súper clase: -	
Sub clase(s): -	
Responsabilidades: getGestorBD() buscarAllBD() buscarAllBDporFiltro() eliminarBD() adicionarBD() modificarBD() comprobarBD()	**Colaboraciones:** GestorADBD BD

Tabla 15. Tarjetas CRC: GestorNCBD

Tarjeta CRC: GestorADBD	
Clase: GestorADBD	
Súper clase: -	
Sub clase(s): -	
Responsabilidades: buscarAllBD() buscarAllBDporFiltro() eliminarBD() adicionarBD() modificarBD() comprobarBD() estadoConexion() conexionBD()	**Colaboraciones:** factoriaEDBD Tbd BD

Tabla 16. Tarjetas CRC: GestorADBD

Para consultar las restantes tarjetas, ver **Anexo 2**.

2.5.2 Patrones de diseño

Un patrón de diseño describe una estructura de diseño que resuelve un problema de diseño en particular dentro de un contexto específico y en medio de "fuerzas" que pueden tener un impacto en la manera que se aplica y utiliza el patrón. Los patrones de diseño son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos, basadas en la experiencia y que se ha demostrado que funcionan (22).

Existen 3 clasificaciones fundamentales de patrones, según GoF (Gang Of Four, en inglés):

De Creación: abstraen el proceso de creación de instancias.

- ✓ **Instancia única / Singleton** para la creación de una instancia de las clases que sean necesarias, como los gestores del negocio y las clases Internacionalización de la herramienta a la hora de inicializar el formulario, para que todos los componentes que lo requieran se encuentren internacionalizados.

Estructurales: se ocupan de cómo clases y objetos son utilizados para componer estructuras de mayor tamaño.

- ✓ Este tipo de patrón no fue necesario utilizarlo en el diseño del sistema.

De Comportamiento: atañen a los algoritmos y a la asignación de responsabilidades entre objetos.

- ✓ **Iterador / Iterator** para proporcionar una forma coherente de acceder secuencialmente a los elementos de una colección, independientemente del tipo de colección subyacente. El patrón se utilizó para recorrer las estructuras de datos provenientes de la ejecución de consultas en las clases factorías del sistema.

Los **patrones GRASP** (Patrones Generales de Software para Asignación de Responsabilidades o General Responsibility Assignment Software Patterns) son otros de los utilizados en el sistema, describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

- ✓ **Creador:** guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea

muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Al escogerlo como creador. Brinda soporte a un bajo acoplamiento lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Este patrón se utiliza en las clases factorías del sistema, donde se crean los objetos provenientes de las consultas a la BD, y los que se registran a través de las entidades del dominio por los usuarios para su persistencia.

- ✓ **Experto:** este patrón permite asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. La responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantendrá encapsulada (disminución del acoplamiento). Utilizado en el gestor de negocio del cliente, ya que este gestor solamente trabaja con la información de las entidades de dominio que representa.
- ✓ **Controlador:** posibilita la asignación de responsabilidades para controlar el flujo de eventos del sistema a clases específicas. Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario enviándolo a las distintas clases según el método llamado. Utilizado en las acciones (clases denotadas por el sufijo “Accion”) para capturar los datos registrados por el usuario en el sistema, y enviarlos a las distintas partes que lo requieran, así como la visualización de los mismos.

2.5.3 Estándares del diseño

Los estándares del diseño son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física de las interfaces; definen un conjunto de reglas para diseñar las interfaces del sistema, los cuales permiten obtener un diseño de alta calidad y que cumpla con las buenas prácticas establecidas en la Ingeniería de Software (23).

A continuación se presentan algunas de las que se utilizarán para el desarrollo del sistema:

- ✓ La separación de los componentes respecto a los bordes del área de trabajo (margen izquierdo,

derecho, superior e inferior) debe ser de 10 píxeles como mínimo.

- ✓ En un formulario, los datos asociados a una misma cosa se agruparán utilizando paneles con título. El título del panel se escribe en formato de oración, en negrita y sin dos puntos al final.
- ✓ Cuando el formulario tiene varios componentes el nombre de las etiquetas se escribe encima de los componentes. El texto en las etiquetas se escribe en formato de oración, sin utilizar negrita y terminado en dos puntos. Los componentes irán debajo, a una distancia de 8 píxeles, cada componente de entrada de datos debe tener una altura de 23 píxeles. La separación horizontal entre componentes debe ser de 20 píxeles y vertical de 15 píxeles. Todo alineado a la izquierda, y en caso de que aparezcan uno debajo del otro se debe tratar de que tengan un mismo tamaño (el del mayor componente).
- ✓ Los JComboBox deben mostrar como texto inicial –Seleccione--. Los ítems para seleccionar deben escribirse en formato de oración y deben aparecer en orden alfabético, a no ser que representen un flujo, en ese caso aparecerían de acuerdo al orden correspondiente. Cuando haya una opción Otros, estará ubicada al final.
- ✓ Cuando haya que introducir una fecha debe ponerse un componente para seleccionar la misma, evitar las entradas manuales del usuario.
- ✓ Los Botones siempre estarán situados en la parte inferior derecha de los formularios, con una altura de 25 píxeles y el ancho depende del Caption, siempre dejando un espacio de 8 píxeles delante y detrás de la palabra o el ícono. Sólo en el caso de los mensajes de confirmación, avisos y error los botones irán en el centro. El texto que contienen debe estar en formato de oración y sin utilizar negrita. Cuando aparezcan varios botones uno al lado del otro, de ser posible todos deben tener el mismo ancho y la separación entre ellos debe ser de 10 píxeles. El ancho mínimo de los botones debe ser de 75 píxeles.

2.5.4 Estándares de codificación

Los estándares de codificación definen un conjunto de reglas para escribir el código, los cuales permiten obtener un código de alta calidad y cumplir con las buenas prácticas establecidas en la Ingeniería de Software. Además posibilita que el software que se obtiene sea fácil de comprender (24). A continuación

se describen los estándares de codificación que se utilizarán para el desarrollo del sistema:

- **Nomenclatura para los paquetes, métodos y clases**

- ✓ Se utilizará la notación Camello en los nombres de las entidades persistentes, entidades de dominio, propiedades y funcionalidades del sistema.
- ✓ Las clases persistentes comenzarán con las siglas definidas en la BD para cada tabla.
- ✓ Las acciones que pertenecen a cada formulario comenzarán con las siglas Accion.
- ✓ Los formularios de tipo popup comienzan con las siglas Frm, y los de tipo panel con las siglas Pnl.
- ✓ Los gestores de negocio comienzan con las siglas GestorNC.
- ✓ Las clases de la lógica de negocio de acceso a datos comienzan con las siglas GestorAD.
- ✓ Las excepciones terminan con el sufijo Exception.

- **Nomenclatura para los componentes de formularios**

- ✓ Los botones (JButton) comienzan con las siglas btn.
- ✓ Los campos de texto (JTextField) comienzan con las siglas txt.
- ✓ Para el componente JDayChooser con la sigla dyc.
- ✓ Las cajas de texto (JComboBox) comienzan con las siglas cmb.
- ✓ Las áreas de texto (JTextArea) comienzan con las siglas txa.
- ✓ Los label (JLabel) comienzan con las siglas lbl.
- ✓ Los cuadros de chequeo (JCheckBox) comienzan con las siglas chb.
- ✓ Los radiobutton (JRadioButton) comienzan con las siglas rbt.
- ✓ Las tablas (JTable) comienzan con las siglas tbl.

2.5.5 Arquitectura en Capas

Es un estilo en el que se divide el sistema en capas. La principal ventaja de estilo consiste en facilitar la gestión de los cambios en un sistema. Sólo basta con trabajar con el nivel que se necesite realizar el

cambio sin tener que tocar el resto de los niveles. Por lo general se divide el sistema en tres capas fundamentales, presentación, lógica de negocio y acceso a datos. Este estilo facilita que cualquier cambio que se realice sobre alguna capa sea transparente respecto a las otras capas que se implementan en el sistema. La abstracción de los desarrolladores de los otros niveles es además otra de las ventajas, estos solo necesitan conocer la interfaz que conecta cada una de las capas. Este estilo arquitectónico fue escogido, ya que permite una gran escalabilidad en los sistemas que se desarrollan y por cumplir todas las características anteriores. Dentro de este estilo se empleó la arquitectura en 4 capas, las cuales se describen a continuación (25):

Capa de presentación: La capa de presentación contiene los formularios con los que interactúan los usuarios finales del sistema y permiten la visualización de toda la información que estos necesitan.

Capa de negocio: Aquí se establecen todas las reglas del negocio que deben ser cumplidas, contiene acciones que gestionan las funcionalidades de los formularios y garantizan la comunicación con los gestores de negocio para la invocación de la lógica de negocio. Además se relaciona con la capa de acceso a datos.

Capa de acceso a datos: La capa de acceso a datos contiene los gestores de acceso a datos y las clases persistentes, estas clases son objetos simples que no implementan lógica y contiene los datos que se van a recuperar o almacenar de la capa de datos, sin perder el paradigma de la programación orientada a objetos.

Capa de datos: Contiene toda la información necesaria para la gestión de las réplicas.

Ventajas

- ✓ Desarrollos paralelos en cada capa.
- ✓ Aplicaciones más robustas debido al encapsulamiento.
- ✓ Mantenimiento y soporte más sencillo.
- ✓ Mayor flexibilidad (se pueden añadir nuevos módulos para dotar al sistema de nueva funcionalidad).
- ✓ Alta escalabilidad. La principal ventaja de una aplicación distribuida bien diseñada es su buen escalado, es decir, que puede manejar muchas peticiones con el mismo rendimiento simplemente añadiendo más hardware. El crecimiento es casi lineal y no es necesario añadir más código para

conseguir esta escalabilidad.

2.5.6 Modelo de Datos

Un modelo de datos es un lenguaje utilizado para la descripción de una base de datos, describe de manera abstracta cómo se representan los datos de una aplicación o sistema de información, tiene gran importancia en el ciclo de desarrollo de software, y de manera particular para la fase de implementación, pues define formalmente las estructuras permitidas y las restricciones que se aplican con el fin de representar los datos del dominio de la aplicación. Está compuesto por objetos: entidades que existen y se manipulan; atributos: características básicas de dichos objetos y relaciones: forma en que se enlazan los objetos entre sí (26).

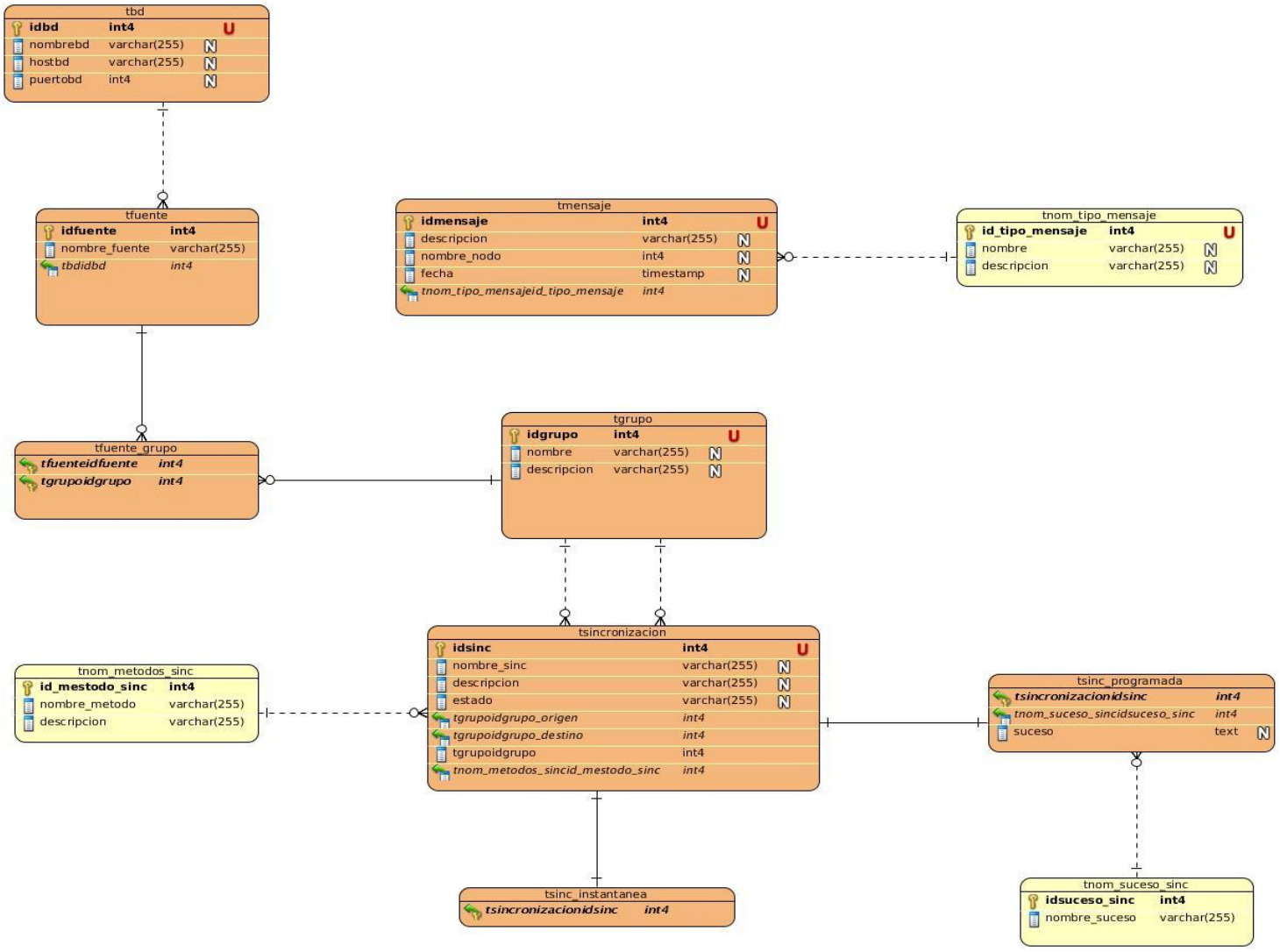


Imagen 1. Modelo de datos

Resumen

Nombre	Descripción
tbd	Almacena las bases de datos que pertenecen al clúster y con las que se establecen las operaciones de réplica.

tfuente	Almacena todas las fuentes que intervienen en el proceso de réplica.
tgrupo	Representa los datos de todos los grupos de réplica que se encuentran en el sistema y pertenecerán a la sincronización.
tfuente_grupo	Representa la relación entre la tabla tfuente y tgrupo, especificando que las fuentes pueden pertenecer a varios grupo y los grupo pueden estar constituidos por varias fuentes.
tsincronizacion	Tabla que almacena los datos de la sincronización.
tsinc_programada	Almacena los tipos de sincronizaciones programadas.
tsinc_instantanea	Almacena los tipos de sincronizaciones instantáneas.
tmensaje	Almacena los mensajes de errores que surgirán en las operaciones transaccionales.
tnom_tipo_mensaje	Nomenclador que representa los tipos de mensajes.
tnom_metodos_sinc	Nomenclador que representa los tipos de métodos que puede tomar una sincronización.
tnom_suceso_sinc	Nomenclador que representa los tipos de sucesos que puede tomar una sincronización programada.

Tabla 17. Resumen modelo de datos

2.5.7 Diagrama de paquetes del diseño

El diagrama de paquetes del diseño refleja cómo el sistema está dividido en agrupaciones lógicas y las relaciones entre estas. Los paquetes están normalmente organizados para maximizar la coherencia interna entre paquetes y minimizar el acoplamiento externo entre los mismos (27). A continuación se

muestra el diagrama de paquetes del diseño del Sistema de Administración y Configuración Chronos, en el que se representan las relaciones entre el paquete “AdminChronos”, el cual se compone por el paquete de “Presentación” donde se encuentran los formularios y las acciones, el paquete “Negocio” que contiene los gestores que sirven de soporte para la lógica del negocio y la comunicación con la capa de acceso a datos y “adAdminChronos” en el que se evidencia la lógica de acceso a datos, y los paquetes de ayuda para la implementación.

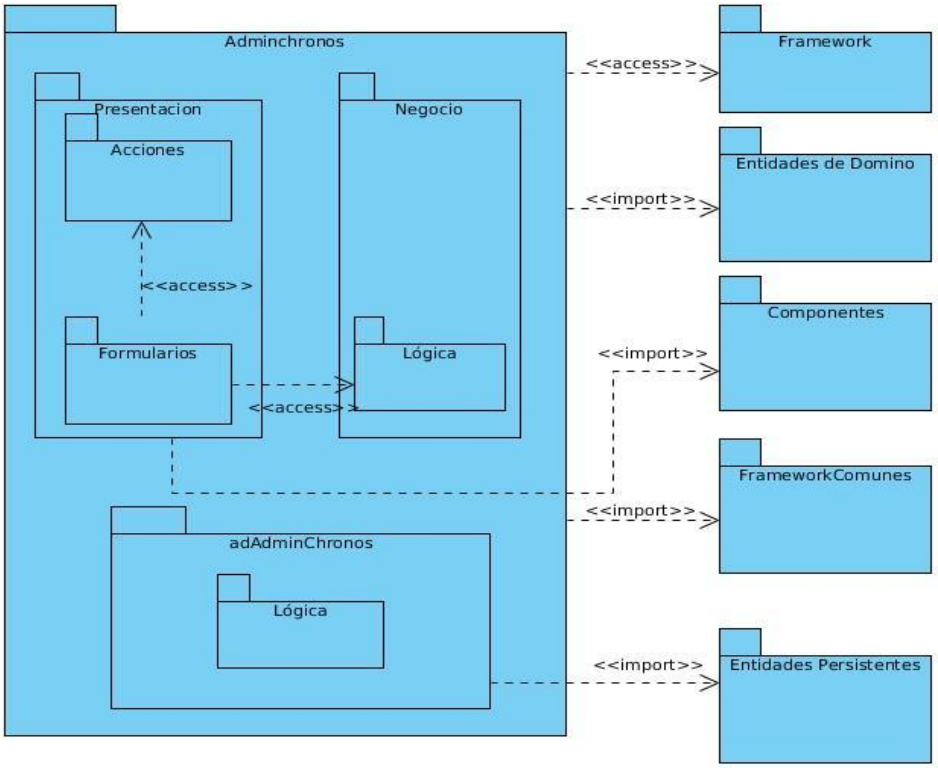


Imagen 2. Diagrama de paquetes

2.5.8 Diagrama de Despliegue

Los Diagramas de Despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria, los que se precisan a través de estereotipos que definen la naturaleza del equipo (28). En este

diagrama se muestra el nodo “Estación de Trabajo”, donde se encontrará la aplicación cliente ejecutándose sobre la JVM, el “Clúster de BD” que contendrá un servidor PostgreSql por cada nodo que lo componga y el “Servidor de BD” en el que estará almacenada la información que se gestione en la aplicación.

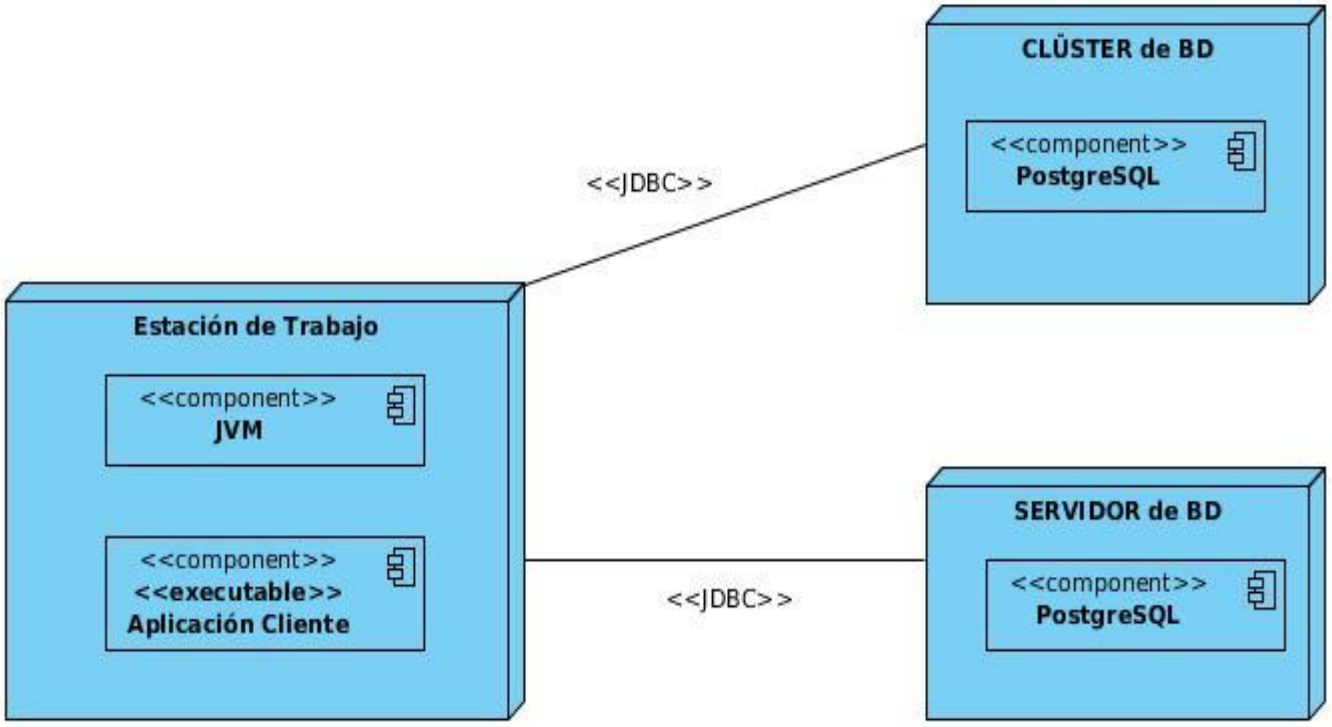


Imagen 4. Diagrama de despliegue

2.6 Conclusiones

En este Capítulo se realizó un estudio de las diferentes características que tendrá el sistema; se determinaron los requisitos funcionales y no funcionales atendiendo a las exigencias del cliente, brindando además una breve descripción de las historias de usuario, las tarjetas CRC, el plan de iteraciones, la duración de cada una de las iteraciones y el plan de entrega de cada funcionalidad, se realizó una propuesta del sistema no funcional, en la que se exponen algunos de los requerimientos con los que contará el sistema, así como los diferentes diagramas que contribuirán a la buena implementación de la herramienta.

Capítulo 3: Implementación y Prueba

3.1 Introducción

En este capítulo se exponen cada uno los componentes que integran el proceso de implementación y prueba de la interfaz de administración y configuración de Chronos. Según las pautas que presenta la metodología de desarrollo XP, el proceso de implementación se caracteriza por ser iterativo e incremental, de modo que se describen cada una de las iteraciones comprobando que se cumplen cada uno de los objetivos planteados en el plan de iteraciones, se exponen las tareas generadas por cada historia de usuario y se realiza el Diagrama de componentes para una mejor visualización de los subsistemas de implementación. Finalmente se procede a realizar conjuntamente con el cliente y el equipo de desarrollo, las pruebas correspondientes a cada historia de usuario, verificando que el sistema cumple todas las funcionalidades.

3.2 Implementación

La implementación es la parte más importante en la programación extrema. XP también propone un modelo de desarrollo colectivo en el que todos los programadores están implicados en todas las tareas. La optimización del código siempre se debe dejar para el final, teniendo en cuenta que hay que hacer que funcione y que sea correcto, más tarde se puede optimizar (17).

La programación extrema propone una forma iterativa para la implementación de un software junto a estas prácticas, da como resultado que al terminar cada iteración se obtenga una versión del producto funcional, éste debe ser probado y mostrado al cliente sirviendo de retroalimentación para el equipo de trabajo.

3.2.1 Tareas de programación o ingenierías

Cada una de estas historias de usuarios se transformará en tareas que serán realizadas por los desarrolladores y programadores, dentro del equipo de desarrollo. Cada tarea de desarrollo corresponderá a un período de uno o tres días de desarrollo (29); a continuación se enumeran cada una de las tareas que se corresponden por las historias de usuario que se obtuvieron en la fase de exploración:

No. Tareas	Tareas por Historias de Usuario
T-1	Adicionar Base de Datos
T-2	Mostrar Base de Datos
T-3	Modificar Base de Datos
T-4	Eliminar Base de Datos
T-5	Adicionar Grupo de réplica
T-6	Mostrar Grupo de réplica
T-7	Modificar Grupo de réplica
T-8	Eliminar Grupo de réplica
T-9	Adicionar Fuentes de réplica
T-10	Mostrar Fuentes de réplica
T-11	Eliminar Fuentes de réplica
T-12	Adicionar Sincronización
T-13	Mostrar Sincronización
T-14	Reconfigurar Sincronización
T-15	Eliminar Sincronización
T-16	Eliminar Mensajes
T-17	Mostrar Mensajes

T-18	Buscar Mensajes
T-19	Administrar Motor de procesamiento de Sentencias (MPS)
T-20	Añadir Filtros de réplica
T-21	Autenticar Usuario

Tabla 18. Definición de tareas de ingeniería

Teniendo como premisa la planificación realizada anteriormente, se llevaran a cabo 3 iteraciones de desarrollo sobre el sistema, permitiendo que al final se logre un producto con todas las restricciones y características deseadas por el cliente. A continuación se detallan cada una de las iteraciones.

3.2.2 Iteración 1

En esta iteración se le da cumplimiento a la implementación de las HU que corresponden a los números 21, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 y 11, consideradas de mayor importancia para el desarrollo de la aplicación, con el fin de obtener una versión del producto con algunas de las funcionalidades críticas como: Autenticar, Gestionar Bases de Datos, Fuentes y Grupos.

Historias de usuario	**Tiempo de implementación(días)**	
	Estimación	**Real**
Autenticar Usuario	2	2
Adicionar Base de Datos	3	3
Mostrar Bases de Datos	1	1
Modificar Base de Datos	2	2

Eliminar Base de Datos	1	1
Adicionar Grupo de réplica	1	1
Mostrar Grupos de réplica	1	1
Modificar Grupo de réplica	1	1
Eliminar Grupo de réplica	1	1
Adicionar Fuentes de réplica	3	3
Mostrar Fuentes de réplica	1	1
Eliminar Fuentes de réplica	3	3

Tabla 19. Iteración 1. Historias de usuario

Tarea de Ingeniería	
No. de la tarea: 21	**No. de la HU:** 21
Nombre de la tarea: Autenticar Usuario	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 09/04/2012	**Fecha fin:** 10/04/2012
Programador responsable: Miguel L. Sojo Hernández	
Descripción: Permite autenticar al usuario, y le añade los diferentes roles a los que puede acceder.	

Tabla 20. Iteración 1. Tareas de ingenierías. Autenticar usuario

Tarea de Ingeniería

No. de la tarea: 1	**No. de la HU:** 1
Nombre de la tarea: Adicionar Base de Datos	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 11/04/2012	**Fecha fin:** 12/04/2012
Programador responsable: Miguel L. Sojo Hernández	
Descripción: Adiciona una Base de Dato, comprobando si la misma existe, según los parámetros que se especificaron.	

Tabla 21. Iteración 1. Tareas de ingenierías. Adicionar base de datos

Tarea de Ingeniería	
No. De la Tarea: 5	**No. De la HU:** 5
Nombre de la tarea: Adicionar Grupo	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 21/04/2012	**Fecha fin:** 22/04/2012
Programador responsable: Miguel L. Sojo Hernández	
Descripción: Se registran y verifican los datos del grupo que se desea adicionar, y se comprueba que no tengan el mismo nombre.	

Tabla 22. Iteración 1. Tareas de ingenierías. Adicionar grupo

Tarea de Ingeniería	
No. de la tarea: 8	**No. de la HU:** 8

Nombre de la tarea: Eliminar Grupo	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 25/04/2012	**Fecha fin:** 26/04/2012
Programador responsable: Miguel L. Sojo Hernández	
Descripción: Si el grupo de réplica a eliminar está implicado en una o más sincronizaciones, no podrá ser eliminado hasta tanto no se eliminen las sincronizaciones a las que pertenece.	

Tabla 23. Iteración 1. Tareas de ingenierías. Eliminar grupo

Para consultar las Tareas de Ingeniería generadas por cada historia de usuario de esta primera iteración, ver **Anexo 3**.

3.2.3 Iteración 2

En esta iteración se implementaron las HU que corresponden a los números 12, 13, 14, 15 y 20. Dichas HU son las que brindan las funcionalidades de Gestionar Sincronizaciones, y añadir los diferentes filtros de réplica que requieran cada una. Pertenecen a esta segunda iteración debido a su complejidad y no a la importancia que representan para la aplicación.

Historias de usuario	**Tiempo de implementación(días)**	
	Estimación	**Real**
Adicionar Sincronización	3	3
Mostrar Sincronización	1	1
Reconfigurar Sincronización	3	3
Eliminar Sincronización	2	2

Añadir Filtros de réplica	3	3

Tabla 24. Iteración 2. Historias de usuario

Tarea de Ingeniería	
No. de la tarea: 12	**No. de la HU:** 12
Nombre de la tarea: Adicionar Sincronización	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 01/05/2012	**Fecha fin:** 03/05/2012
Programador responsable: Miguel L. Sojo Hernández	
Descripción: Se registran y verifican los datos por lo que está compuesta la sincronización.	

Tabla 25. Iteración 2. Tareas de ingenierías. Adicionar sincronización

Para consultar las Tareas de Ingeniería generadas por cada historia de usuario en esta segunda iteración, ver **Anexo 4**.

3.2.4 Iteración 3

En esta última iteración, se le dio cumplimiento a las historias de usuario de media y baja prioridad (16, 17, 18 y 19), brindando una versión final del producto listo para pasar a ser probado, y posteriormente ponerlo en funcionamiento.

Historias de usuario	**Tiempo de implementación(días)**	
	Estimación	**Real**
Eliminar Mensajes	1	1

Mostrar Mensajes	1	1
Buscar Mensajes	1	1
Administrar Motor de procesamiento de Sentencias (MPS)	3	3

Tabla 26. Iteración 3. Historias de usuario

Tarea de Ingeniería	
No. de la tarea: 17	**No. de la HU:** 17
Nombre de la tarea: Mostrar Mensajes	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 23/05/2012	**Fecha fin:** 24/05/2012
Programador responsable: Miguel L. Sojo Hernández	
Descripción: Se muestran los datos de los diferentes tipos de mensajes de errores que genera la aplicación.	

Tabla 27. Iteración 2. Tareas de ingenierías. Mostrar mensajes

Para consultar las Tareas de Ingeniería generadas por cada historia de usuario en esta tercera iteración, ver **Anexo 5**.

3.2.5 Diagrama de componentes

Los diagramas de componentes son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre los elementos de implementación (30). A continuación se muestra el diagrama de componentes del Sistema de Administración y Configuración de

Chronos, los componentes que lo integran son las acciones que se encuentran en el paquete “Acciones” y se encargan de controlar el flujo de eventos que proviene de los “Formularios”, recibir los datos que registra el usuario y enviarlos a los distintos gestores “Negocio” según la lógica requerida en cada acción, y que estos se encarguen de comunicarse con el paquete “Acceso Datos” para a través de los gestores de acceso a datos gestionar toda la información que necesite el usuario, apoyándose en las entidades de dominio, entidades persistentes, librerías y en los servicios que brindan los paquetes “i18n” (para internacionalizar el sistema), “frameworkComunes” y “framework” (que se responsabilizan de gestionar el marco de trabajo de la aplicación):

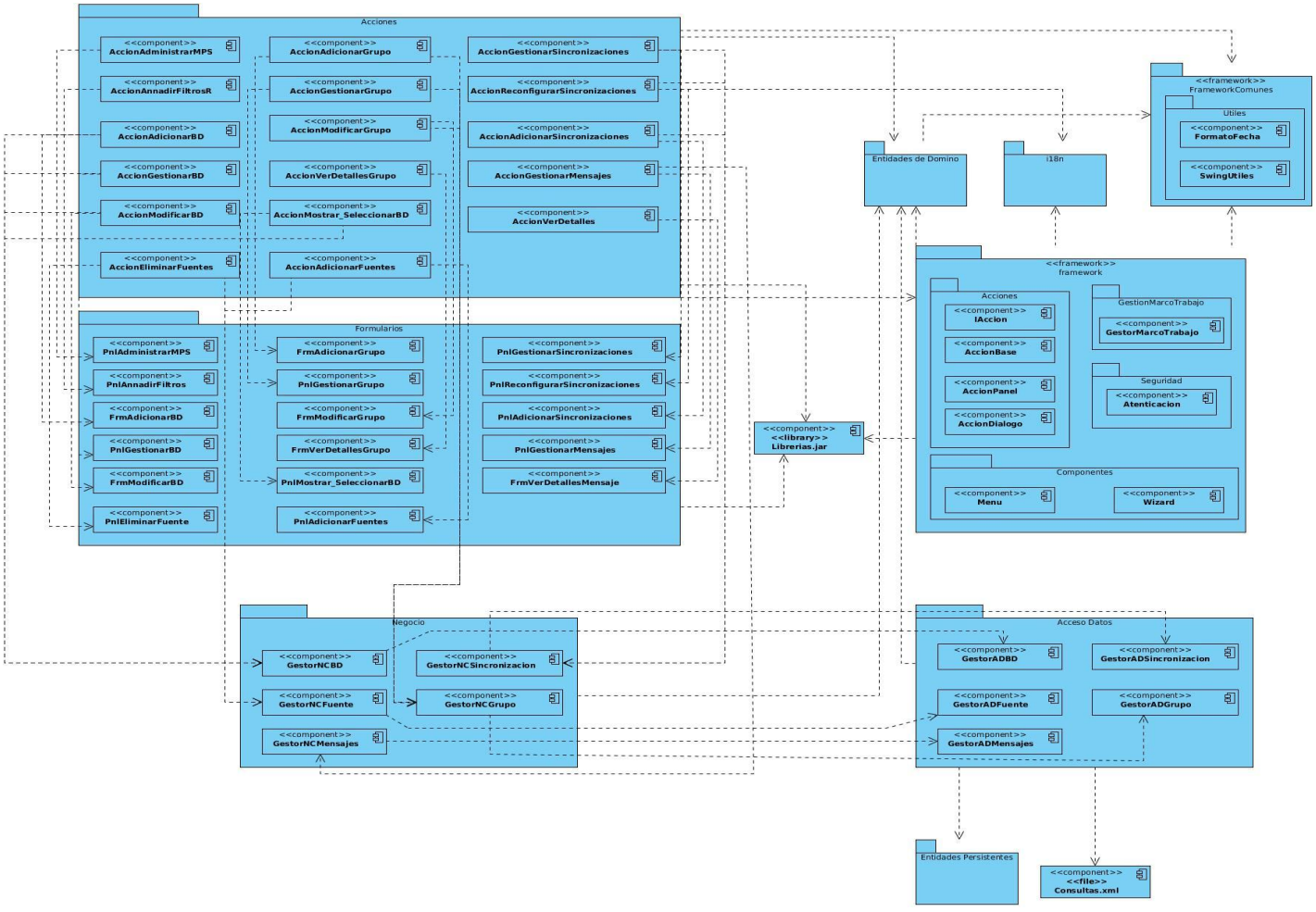


Imagen 3. Diagrama de componentes

3.3 Prueba

Uno de los pilares de XP es el proceso de pruebas. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. Todo esto contribuye a elevar la calidad de los productos desarrollados y a la seguridad de los programadores a la hora de introducir cambios o modificaciones (17).

La metodología XP divide las pruebas en dos grupos: pruebas unitarias, desarrolladas por los programadores, encargadas de verificar el código de forma automática y las pruebas de aceptación, destinadas a evaluar si al final de una iteración se obtuvo la funcionalidad requerida, además de comprobar que dicha funcionalidad sea la esperada por el cliente, estas pruebas son diseñadas por el cliente.

3.3.1 Pruebas unitarias

Las pruebas unitarias son creadas por los propios desarrolladores para comprobar si el código realiza lo que se espera de él de forma automática y rápida. Una prueba unitaria consiste en probar una porción de código en todos los escenarios posibles a los que se pueda enfrentar su uso en la aplicación (31).

Para darle cumplimiento a las mismas se determinó realizar la prueba del camino básico, para detectar errores debido a cálculos incorrectos, comparaciones incorrectas o flujos de control inapropiados. Las pruebas del camino básico y de bucles son técnicas muy efectivas para descubrir una gran cantidad de errores en los caminos. El método del camino básico permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida corno guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Se diseñó el grafo de flujo, para determinar la complejidad ciclomática del método buscarAllBD() de la clase GestorADBD y obtener el número de caminos independientes del conjunto básico del método, al igual que el límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez (31).

```
public List<BD> buscarAllBD(){

    List<BD> listaDatos = new ArrayList<BD>();
    eManager.getTransaction().begin();
    Query consulta = eManager.createQuery("select t from Tbd t");
    List<Tbd> listaBD = consulta.getResultList();
    eManager.getTransaction().commit();
    eManager.close();
    if (listaBD.isEmpty()) {
        return null;
    }
    listaDatos = factoria.convertirLista(listaBD);
    for (int i = 0; i < listaDatos.size(); i++) {

        try {
            Class.forName("org.postgresql.Driver");
            String direccion="jdbc:postgresql://"+listaDatos.get(i).getHostbd()+":"+listaDatos.get(i).getPuertobd()+"/"+listaDatos.get(i).getNombrebc
            DriverManager.setLoginTimeout(1);
            conexion = DriverManager.getConnection(direccion,"postgres","postgres");
            if(conexion != null){listaDatos.get(i).setConexion(true);}
            else{listaDatos.get(i).setConexion(false);}
        } catch (Exception ex) {
            listaDatos.get(i).setConexion(false);
        }
    }
    return listaDatos;
}
```

Imagen 5. Método buscarAllBD

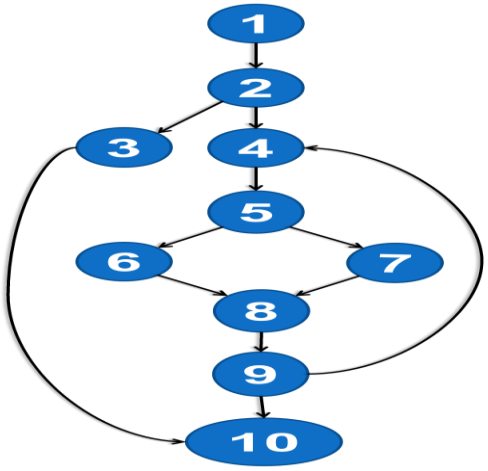


Imagen 6. Grafo de flujo: buscarAllBD

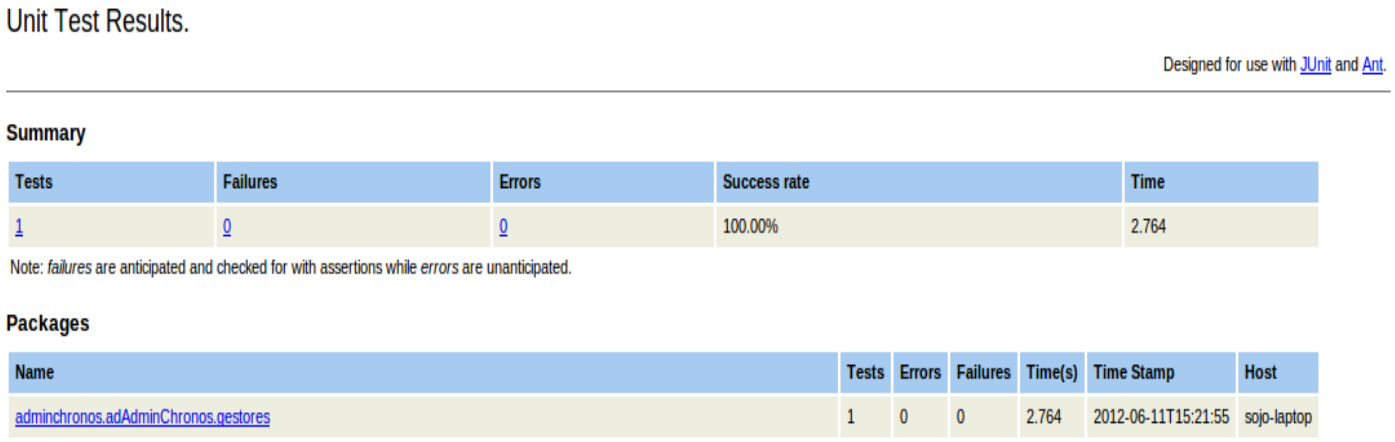
La complejidad ciclomática V (G), del grafo de flujo anterior se define como.

V (G)=A-N+2, donde A es el número de aristas del grafo de flujo y N es el número de nodos del mismo.

V (G)= 12 – 10 + 2.

V (G)= 4.

Se determinó la complejidad ciclomática del grafo de flujo resultante, obteniéndose la realización de 4 casos de prueba para asegurar que por cada camino independiente no existan errores. Para ver los resultados de los restantes casos de prueba, ver **Anexo 6.** Además, se realizaron las pruebas unitarias a las funcionalidades de mayor complejidad, atendiendo al desarrollo de estas en cada iteración, y determinadas por su valor para que el sistema cumpla con los requisitos funcionales. Obteniéndose en la primera iteración un 30% de funcionalidades correctas, en la segunda iteración un 80% de funcionalidades correctas y en la tercera iteración un 100% de funcionalidades correctas, logrando que el sistema cumpla con los requisitos del cliente.



Unit Test Results.

Designed for use with JUnit and Ant.

Summary

Tests	Failures	Errors	Success rate	Time
1	0	0	100.00%	2.764

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

Packages

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
adminchronos.adAdminChronos.gestores	1	0	0	2.764	2012-06-11T15:21:55	sojo-laptop

Imagen 7. Resultados. Prueba unitaria: buscarAllBD

3.3.2 Pruebas de aceptación

Las pruebas de aceptación se crean a partir de las historias de usuario en cada ciclo de iteración. Una historia de usuario puede tener todas las pruebas de aceptación que necesite para asegurar su correcto funcionamiento. En ellas se especifican, desde la perspectiva del cliente, los escenarios para probar que una historia de usuario ha sido implementada correctamente. El objetivo final de estas es garantizar que los requisitos han sido cumplidos y que el sistema es aceptable. Una historia de usuario no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación (31).

A continuación se describen los casos de pruebas de aceptación realizados a las principales

funcionalidades en la iteración 1 y 2. Para consultar los restantes casos de pruebas, ver **Anexo 7**.

Casos de Prueba de Aceptación	
Código: HU21_P1	**No. Historia de usuario:** 21
Nombre: Autenticar Usuario	
Descripción: Prueba de funcionalidad de la autenticación del usuario.	
Condiciones de ejecución: Los usuarios que tienen acceso al sistema deben estar creados y configurados con sus respectivos permisos.	
Entradas / Pasos de ejecución: El usuario procede a la autenticación en el sistema.	
Resultado esperado: Una vez autenticado correctamente, el usuario accede al sistema y a la información que este tendrá acceso en dependencia del rol que tenga (administrador).	
Evaluación de la prueba: Prueba Satisfactoria	

Tabla 28. Caso de prueba. Autenticar usuario

Caso de prueba: Autenticar Usuario			**No. Caso de Prueba:** 1
Clases válidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**

Introducción correcta de las credenciales del usuario: Usuario: administrador Contraseña: **********	Una vez entrados los datos del usuario que intenta acceder al sistema, al presionar en el botón aceptar, se mostrará la página principal de la aplicación con las opciones que el usuario tendrá acceso en dependencia de su rol.	Satisfactorio	La aplicación contará con un solo usuario: **administrador**.
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
Introducción incorrecta de las credenciales del usuario: Usuario: vacío Contraseña: vacío	Muestra el mensaje de error:	Satisfactorio	
Evaluación de la prueba: Satisfactoria			

Tabla 29. Validación. Autenticar usuario

Casos de Prueba de Aceptación	
Código: HU1_P2	**No. Historia de usuario:** 1
Nombre: Adicionar Base de Datos	
Descripción: Prueba de funcionalidad de adicionar una Base de datos (BD) al Sistema.	
Condiciones de ejecución: El usuario debe estar previamente autenticado.	

Entradas / Pasos de ejecución: El usuario registra los datos específicos de la BD que desea añadir, y presiona la opción adicionar.
Resultado esperado: Se muestra un mensaje: “La base de datos se ha adicionado correctamente”.
Evaluación de la prueba: Prueba Satisfactoria

Tabla 30. Caso de prueba. Adicionar base de datos

Caso de prueba: Adicionar Base de Datos			**No. Caso de Prueba:** 2
Clases válidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
El usuario inserta los datos de la BD que va a adicionar.	El sistema muestra un mensaje: “La base de datos se ha adicionado correctamente”.	Satisfactorio	El usuario puede optar por la opción “Comprobar”, para asegurarse de los datos introducidos.
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
El usuario deja campos vacíos al insertar los datos de la BD que desea añadir.	El sistema muestra un mensaje: “Todos los campos son obligatorios”.	Satisfactorio	
Evaluación de la prueba: Satisfactoria			

Tabla 31. Validación. Adicionar Base de datos

Casos de Prueba de Aceptación	
Código: HU12_P3	**No. Historia de usuario:** 12

Nombre: Adicionar Sincronización
Descripción: Prueba de funcionalidad de adicionar una Sincronización al Sistema.
Condiciones de ejecución: El usuario debe estar previamente autenticado; Deben existir más de dos grupos que cumplan con las características de réplica.
Entradas / Pasos de ejecución: El usuario registra los datos específicos de la Sincronización que desea añadir, en caso de escoger el tipo de sincronización programada debe añadir los datos de la frecuencia de réplica, presiona la opción adicionar.
Resultado esperado: Se muestra un mensaje: "La Sincronización se ha adicionado correctamente".
Evaluación de la prueba: Prueba Satisfactoria

Tabla 32. Caso de prueba. Adicionar sincronización

Caso de Prueba: Adicionar Sincronización			**No. Caso de Prueba:** 3
Clases válidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
El usuario inserta los datos de la Sincronización que va a adicionar.	El sistema muestra un mensaje: "La Sincronización se ha adicionado correctamente".	Satisfactorio	
Clases inválidas	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
El usuario deja campos vacíos al insertar los datos de la Sincronización que desea añadir.	El sistema muestra un mensaje: "Todos los campos son obligatorios".	Satisfactorio	

Evaluación de la prueba: Satisfactoria

Tabla 33. Validación. Adicionar sincronización

3.3.3 Resultados de las pruebas

Durante la etapa de prueba el sistema fue sometido a una primera iteración, realizándose 14 casos de prueba y obteniéndose 10 no conformidades para un 28,57% de resultados satisfactorios, luego de corregir los errores encontrados se realizó una segunda iteración con un total de 17 casos de prueba en la cual se registraron 6 no conformidades para un 64,70% de resultados satisfactorios, se solucionaron las mismas, y se determinó realizar una tercera iteración, donde no se encontraron errores en 14 casos de prueba realizados para un 100% de resultados satisfactorios, lo que permite que el sistema esté listo para pasar por el departamento de calidad del centro. Para una mejor compresión se muestra la siguiente gráfica, en la que se muestra las no conformidades y pruebas satisfactorias en cada iteración.

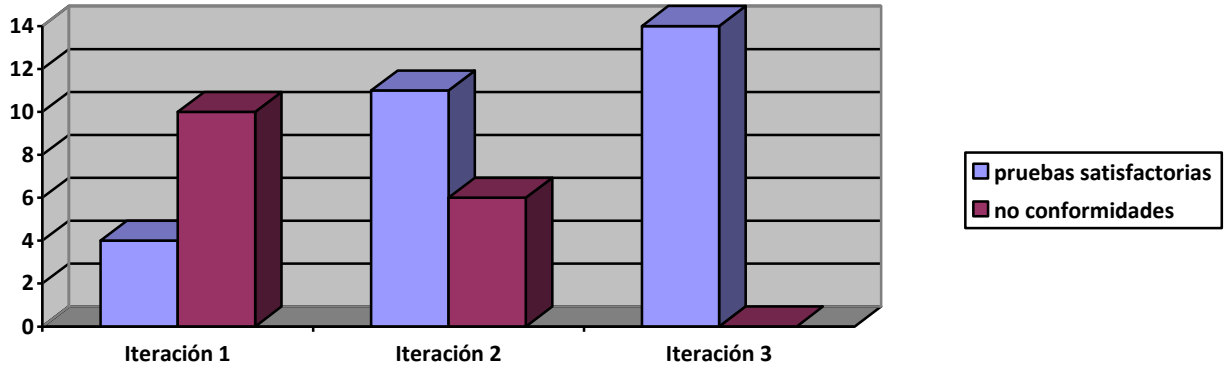


Imagen 8. Resultados generales de las pruebas de aceptación

3.4 Conclusiones parciales

En este capítulo se evidenciaron las fases de implementación y prueba, se conformaron las distintas tareas de ingeniería que le dan solución a las historias de usuarios que se determinaron, se llevaron a cabo las 3 iteraciones, implementando en cada una, las historias de usuario que les correspondían según el plan previsto, así como las pruebas de aceptación a cada una de estas, con el fin de validar la calidad

de la aplicación, y que el cliente corroborara las funcionalidades de la misma. Al finalizar este capítulo se concluye todo el proceso de desarrollo que ocupa a esta investigación.

Conclusiones Generales

Producto de la investigación que se llevó acabo, se creó una aplicación de escritorio, para la administración y configuraciones de réplica del Sistema Chronos. Los objetivos planteados fueron cumplidos de manera satisfactoria obteniendo a su vez una serie de resultados:

- ✓ Se elaboró el marco teórico de la investigación.
- ✓ Se obtuvieron los elementos del diseño de la solución.
- ✓ Se realizó satisfactoriamente la implementación de los elementos de diseño obtenidos.
- ✓ Se validó la solución propuesta garantizando la calidad de la misma.

Recomendaciones

A partir del estudio y la investigación realizada para llevar a buen término el presente trabajo se recomienda:

- ✓ Aplicar las experiencias obtenidas en el desarrollo de aplicaciones similares.
- ✓ Continuar el proceso de validación del sistema para la obtención de la certificación de calidad del Grupo de Calidad UCI.
- ✓ Trabajar en nuevas versiones del sistema que permitan la mejora continua del mismo a partir de nuevos requisitos del cliente.
- ✓ Incluir en las nuevas versiones la utilización del patrón Fachada para proporcionar una interfaz entre los gestores clientes y de acceso a datos, sin especificar sus clases concretas, y lograr un bajo acoplamiento entre estos.

Referencias Bibliográficas

1. **León, Rolando Alfredo Hernández y Gonzáles, Sayda Coello.** *EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN CIENTIFICA.* Ciudad Habana : EDUNIV Editorial Universitaria, 2002. Vol. DISEÑO METODOLÓGICO DE LA INVESTIGACIÓN CIENTÍFICA. ISBN: 959-16-0343-6.

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

3. **Basurto, Maria de los Ángeles Merino.** slideshare. *slideshare.* [En línea] 25 de Marzo de 2011. [Citado el: 21 de Febrero de 2012.] http://www.slideshare.net/LI-Angeles/introduccin-a-las-bases-de-datos-parte1.

4. CAVSI. *CAVSI.* [Citado el: 21 de Febrero de 2012.] http://www.cavsi.com/preguntasrespuestas/que-es-un-sistema-gestor-de-bases-de-datos-o-sgbd/.

5. **Sulistio, Anthony, Yeo, Chee Shin y Buyya, Rajkumar.** Cluster computing at a glance. *Software Practice and Experience.* 1999, págs. 551-576.

6. **Date, C. J.** *Introducción a los Sistemas de Bases de Datos.* s.l. : Pearson Prentice Hall, 2006. pág. 480.

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 : s.n., 2012.

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

9. **José H. Canós, Patricio Letelier y Penadés, Mª Carmen.** *Métodologías Ágiles en el Desarrollo de Software.* Valencia : s.n. http://www.willydev.net/descargas/prev/TodoAgil.Pdf.

10. **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.

11. **José H. Canós, Patricio Letelier y Penadés, Mª Carmen.** *Métodologías Ágiles en el Desarrollo de Software.* Valencia : s.n. http://www.willydev.net/descargas/prev/TodoAgil.Pdf.

12. Buenas Tareas. *Buenas Tareas.* [Citado el: 01 de Marzo de 2012.] http://www.buenastareas.com/ensayos/Uml-Lenguaje-Unificado/3898112.html.

13. **Aguinaga, David Sande y Saúl.** Slideshare. *Slideshare.* [Citado el: 23 de Febrero de 2012.] http://www.slideshare.net/davidsande/presentacion-herramientas-case.

14. Free Download Manager. *Free Download Manager.* 05 de Marzo de 2007. [Citado el: 01 de Marzo de 2012.]http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28Iglesia_Anglicana%29_%5BMac_OS_X_cuenta_14717_p/.

15. **García-Beltrán, A. y Arranz, J.M.** Open Course Ware. *Open Course Ware.* [En línea] 25 de Febrero de 2010. [Citado el: 03 de Marzo de 2012.] http://ocw.upm.es/lenguajes-y-sistemas-informaticos/programacion-en-java-i/Contenidos/LecturaObligatoria/1-introduccion.pdf.

16. **Latorre, Galo.** Programacion-ii. *Programacion-ii.* [En línea] 24 de Maro de 2010. [Citado el: 03 de Mayo de 2012.] http://gl-epn-programacion-ii.blogspot.com/2010/03/jvm-jdk-jre-conceptos-fundamentales-de.html.

17. **Castillo Oswaldo, Figueroa Daniel, Sevilla Héctor.** Programación Extrema. [Citado el: 2012 de Mayo de 10.] http://programacionextrema.tripod.com/fases.htm.

18. **Beck, Kent, Jeffries, Ron y Cinningham, Ward.** Ecured. *Ecured.* [En línea] [Citado el: 02 de Mayo de 2012.] http://www.ecured.cu/index.php/Programaci%C3%B3n_Extrema_o_XP.

19. **Reynoso, Carlos.** Scribd. *Scribd.* [En línea] Abril de 2004. [Citado el: 05 de Mayo de 2012.] http://es.scribd.com/doc/72837716/5/eXtreme-Programming-XP.

20. [En línea] [Citado el: 03 de Mayo de 2012.] www.inf.utfsm.cl/~visconti/xp/Plan_Entregas_2.doc.

21. **Castillo Oswaldo, Figueroa Daniel, Sevilla Héctor.** Programación Extrema. [Citado el: 2012 de Mayo de 10.] http://programacionextrema.tripod.com/fases.htm.

22. **Pressman, Roger S.** *Ingeniería del Software. Un enfoque práctico Sexta Edición.*

23. **Proyecto SIGESAP.** Documento Estándares de Diseño.

24. **Proyecto SIGESAP.** Documento Estándares de Codificación.

25. Slideshare. *Slideshare.* [En línea] [Citado el: 05 de Mayo de 2012.] http://www.slideshare.net/Decimo/arquitectura-3-capas.

26. MSDN. *MSDN.* [En línea] Microsoft. [Citado el: 05 de Mayo de 2012.] http://msdn.microsoft.com/es-es/library/aa291591%28v=vs.71%29.aspx.

27. 4Boot. [En línea] [Citado el: 10 de Mayo de 2012.] http://alexcertz.blogspot.com/2012/01/ensayo-de-diagrama-de-paquetes-y.html.

28. **Susana, Quisbert Limachi Nancy y Michael, Marca Huallpara Hugo.** Diagrama de Despliegue. *Análisis y Diseño.* [En línea] virtual.usalesiana.edu.bo/web/practica/archiv/despliegue.doc.

29. **Gutiérrez, J. J., y otros, y otros.** Lenguajes y Sistemas informáticos. [En línea] [Citado el: 11 de Mayo de 2012.] http://www.lsi.us.es/~javierj/investigacion_ficheros/PSISEXTREMA.pdf.

30. **Jacobson, I.; Booch, G. y Rumbaugh, J.**; "El Proceso Unificado de Desarrollo de software". 2000.

31. **Pressman, Roger S.** *Ingeniería del Software. Un enfoque práctico. Quinta Edición, Pruebas.* Madrid : s.n., 2001.

Bibliografía

[1] Rolando Alfredo Hernández León y Sayda Coello González, *EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN CIENTIFICA*. Ciudad de la Habana: EDUNIV Editorial Universitaria, 2002.

[2] «Object Management Group - UML». [Online]. Available: http://www.uml.org/. [Accessed: 09-jun-2012].

[3] «UML, BPMN and Database Tool for Software Development». [Online]. Available: http://www.visual-paradigm.com/. [Accessed: 09-jun-2012].

[4] «PostgreSQL: The world's most advanced open source database». [Online]. Available: http://www.postgresql.org/. [Accessed: 09-jun-2012].

[5] Mike Keith y Merrick Schincariol, *Pro JPA 2 Mastering the Java™ Persistence API*. Estados Unidos: .

[6] «Welcome to NetBeans». [Online]. Available: http://netbeans.org/. [Accessed: 09-jun-2012].

[7] «Welcome to JUnit.org! | JUnit.org». [Online]. Available: http://www.junit.org/. [Accessed: 11-jun-2012].

Glosario de Términos

A

API: La interfaz de programación de aplicaciones (API) es el conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

AWT: Kit de Herramientas de Ventana Abstracta (Abstract Window Toolkit, AWT por sus siglas en Inglés) es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java. AWT es ahora parte de las Java Foundation Classes (JFC); la API estándar para suministrar una interfaz gráfica de usuario (GUI) para un programa Java.

B

Bytecode: es el código intermedio entre el código fuente y el código máquina. Suele tratársele como un fichero binario que contiene un programa ejecutable similar a un módulo objeto.

C

Clúster: Es un grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio. De un sistema de este tipo se espera que presente combinaciones de los siguientes servicios:

- ✓ Alto rendimiento.
- ✓ Alta disponibilidad.
- ✓ Equilibrio de carga.
- ✓ Escalabilidad.

E

EntityManager: Con la llegada de EJB 3 y JPA nace la figura del EntityManager para simplificar la persistencia de objetos.

Esclavo: En la replicación de bases de datos los servidores Slave (esclavo) leen las consultas almacenadas en el log binario del servidor Master y las ejecutan localmente, de esta manera mantienen

una copia exacta de los datos almacenados en el Master.

F

Framework Es una estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje de scripting entre otras aplicaciones para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

H

Hibernate: Es una herramienta de mapeo objeto-relacional 38 para la plataforma Java. Facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos XML. Hibernate busca solucionar el problema de la diferencia entre estos dos modelos: el usado en la memoria de la computadora (orientación a objetos) y el usado en las bases de datos (modelo relacional).

I

Interfaz: En informática, una interfaz es la parte del programa informático que permite el flujo de información entre varias aplicaciones o entre el propio programa y el usuario. En software también se habla de interfaz gráfica de usuario, que es un método para facilitar la interacción del usuario con el ordenador o la computadora a través de la utilización de un conjunto de imágenes y objetos pictóricos (iconos, ventanas, formularios, etc.).

J

Java: Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los 90.

JPA: Java Persistence API, más conocida por su sigla JPA, es la API de persistencia desarrollada para la plataforma Java EE e incluida en el estándar EJB3. Esta API busca unificar la manera en que funcionan las utilidades que proveen un mapeo objeto-relacional.

J2EE Java 2 Enterprise Edition. Es una parte de la plataforma de programación Java. Basada en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

J2SE Java 2 Standard Edition. Es una parte de la plataforma de programación Java.

JVM: (Java Virtual Machine o Máquina Virtual Java). Es un programa nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial.

L

Librerías dinámicas: Ficheros independientes que pueden ser invocados desde cualquier ejecutable, de modo que su funcionalidad puede ser compartida por varios ejecutables.

M

Maestro: En la replicación de bases de datos en el servidor Master (maestro o amo) es donde se realizan todas las consultas de modificación de datos.

Maestro-Esclavo: Un Esclavo puede tener un único maestro. Un maestro puede tener múltiples esclavos. Los maestros envían las modificaciones realizadas en las BD a los esclavos. Los esclavos no pueden modificar la BD a su antojo, solamente recibir las modificaciones del maestro y aplicarlas. En resumidas cuentas, como máximo un maestro (escritura) y múltiples esclavos (lectura).

Multi-Maestro: La replicación multi-maestro también conocida peer to peer está compuesta de múltiples sitios maestros que participan igualmente en un modelo que soporta actualizaciones desde cualquier sitio. Las actualizaciones a un sitio individual son propagadas a todos los demás sitios maestros que participan.

Mapeo Objeto-Relacional: Más conocido por su nombre en inglés, Object-Relational-Mapping, o sus siglas ORM, es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia.

N

Nodo: En términos de la propuesta para replicar base de datos fragmentados, son bases de datos PostgreSql que intervienen en la réplica.

P

Plataformas

Entorno de software común en el cual se desenvuelve la programación de un grupo definido de aplicaciones.

POJO: (acrónimo de Plain Old Java Object) es una sigla creada por Martin Fowler, Rebecca Parsons y

Josh MacKenzie, utilizada por programadores en Java para enfatizar el uso de clases simples y que no dependen de un framework en especial.

S

Scripts: Un conjunto de comandos escritos en un lenguaje interpretado.

T

Trigger(o disparador): En una Base de datos, es un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación de inserción (INSERT), actualización (UPDATE) o borrado (DELETE).

U

UNIX: Sistema operativo portable, flexible, potente, con entorno programable, multiusuario y multitarea, muy difundido.

Anexos

Anexo 1: Historias de usuario.

Anexo 2: Tarjetas CRC.

Anexo 3: Tareas de ingeniería.

Tareas de ingeniería para la primera iteración.

Anexo 4: Tareas de ingeniería.

Tareas de ingeniería para la segunda iteración.

Anexo 5: Tareas de ingeniería.

Tareas de ingeniería para la tercera iteración.

Anexo 6: Casos de pruebas unitarias.

Anexo 7: Casos de pruebas de aceptación.