UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Centro de Consultoría y Desarrollo de Arquitecturas Empresariales

Mecanismo de captura de cambios de estructura en bases de datos gestionadas en PostgreSQL utilizando el Replicador de datos Reko

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

Autor: Liosvel Medina Molina
Tutores: Ing. Nayibi Martín Peña
Ing. Mario Romero González

La Habana

"Año 59 de la Revolución"

Agradecimientos

Le agradezco a Dios por responder a mis oraciones y permitir que todo esto fuera posible.

A mi familia, por su esfuerzo, su dedicación sin reservas y por apoyar mi decisión de ser ingeniero.

A mi familia en Cristo por sus oraciones, consejos y por su ayuda, en especial a Aurora y Elizabeth.

A mis amigos Alejandro, Dayana, Victor, Yaima y Alfredo, por compartir mi sueño.

A mis profesores, compañeros de grupo, de apartamento y al personal de la dieta, por ser mi otra familia y acompañarme durante estos últimos años.

A mi tutora Nayibi, por su experiencia, su paciencia, su comprensión y su esfuerzo en ayudarme a salir adelante como ingeniero.

Dedicatoria

Dedico este trabajo a mis padres por su amor incondicional y por su ejemplo, por estar presentes en la buenas para compartir y en las malas para consolar; celebrado mis victorias y llorando mis derrotas. Esos que no se pueden reemplazar y sólo se tienen una vez en la vida. A ellos que me ayudaron a cumplir mis sueños les debo todo lo que tengo y lo que soy.

Los quiero mucho.

Declaración Jurada de Autoría y Agradecimientos

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

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

_____	_____
Firma del Autor	Firma del Tutor

Resumen

Los Sistemas de Bases de Datos Distribuidos, dependen de procesos de réplica de datos entre sus nodos para garantizar la integridad de sus datos. Si la estructura de la Base de Datos origen difiere a la del destino durante la réplica de datos, se producen conflictos; que pueden ser evitados realizando réplicas de estructura entre dichas Bases de Datos.

En la presente investigación se pretende mejorar el Replicador de Datos Reko, desarrollado en la Universidad de las Ciencias Informáticas (UCI) por el Centro de Consultoría y Desarrollo de Arquitecturas Empresariales (CDAE); ya que poseía un mecanismo que no evitaba completamente los conflictos, pues sólo capturaba los cambios de estructura más importantes en Bases de Datos gestionadas en PostgreSQL y con un alto consumo de recursos del sistema. El presente trabajo, propone un nuevo mecanismo de captura de cambios de estructura mediante *triggers DDL*, en bases de datos gestionadas en PostgreSQL, que contribuye a evitar conflictos en la réplica de datos y reducir el uso de recursos del sistema.

Se define la metodología Proceso Unificado Ágil (Variación UCI) que guía el desarrollo de la solución propuesta, así como las herramientas y tecnologías empleadas, las cuales son de código abierto y de licencias gratuitas. Se describe el diseño del sistema, la implementación de las funcionalidades y el estándar de codificación utilizado. Finalmente se valida el nuevo programa mediante la aplicación de métricas de software, la realización de pruebas y el análisis del consumo de recursos antes y después de la mejora.

Palabras claves: Sistemas de Bases de Datos Distribuidos, réplica de estructura, *triggers DDL*.

Índice

Índice de Tablas

Índice de Figuras

Introducción

El acelerado avance de las tecnologías modernas vinculado al desarrollo de la Informática, ha provocado la creciente demanda del ser humano de almacenar grandes volúmenes de información, surgiendo así las Bases de Datos (BD), que permiten organizar, clasificar y procesar la información guardada para su posterior análisis y uso, e incluso ser consultada por agentes externos.

La expansión global de los negocios y la descentralización regional de las operaciones de múltiples empresas e industrias, provocaron la necesidad de compartir información a nivel mundial, dando origen así a los Sistemas de Base de Datos Distribuidas (SBDD). Los cuales están compuestos por nodos[1] separados geográficamente sin límite de distancia y se comunican entre sí mediante la red, permitiendo el acceso a la información como si se tratase de una única BD y aumentando el rendimiento en general, pues las transacciones pueden ocurrir en varios nodos simultáneamente.

Para mantener su información actualizada y con ello conservar la integridad estructural, homogeneidad y disponibilidad de sus datos, un SBDD utiliza principalmente procesos de réplica, que consisten en analizar los cambios ocurridos en cada nodo y luego aplicar dichos cambios donde se requiera en el resto del sistema. De esta forma se mantiene una sincronía total o parcial de los datos, lo cual es una fuerte estrategia ante desastres naturales o fallos técnicos, pues al afectarse un nodo específico, la información perdida estará respaldada en otra ubicación.

Durante el proceso de réplica de datos pueden ocurrir eventos no deseados, que indican que la acción no se completó correcta o completamente; entre ellos se encuentran los conflictos de réplica de datos, comunes en los SBDD y provocados por diversas razones como errores durante la introducción de la información, errores a causa de diferencias entre las estructuras de los nodos origen y destino u otras causas que surgen de la desigualdad de la información existente en las bases de datos. Algunos de estos conflictos se evitan utilizando procesos de réplica de estructura, los cuales consisten en detectar cambios estructurales en cada nodo, para posteriormente aplicarlos en los nodos destino.

[1] Nodo: Consiste en una BD alojada en un servidor, u otro SBDD.

En la Universidad de las Ciencias Informáticas (UCI), en el Centro de Consultoría y Desarrollo de Arquitecturas Empresariales (CDAE), se desarrolla el Replicador de Datos Reko, surgido en el año 2007 a partir de la necesidad presentada por el Sistema de Gestión Penitenciario Venezolano (SIGEP). Reko cubre las principales necesidades de la distribución de datos entre los Sistemas Gestores de Bases de Datos (SGBD) más populares como PostgreSQL, MySQL, Microsoft SQL Server y Oracle, en la recuperación, protección, sincronización, transferencia entre diversas localizaciones y la centralización de los datos en una única localización.

Actualmente, Reko posee un mecanismo que captura algunos cambios de estructura utilizando funciones creadas en PostgreSQL, que se ejecutan a partir de tareas programadas desde Reko. Este mecanismo no evita completamente los conflictos, pues no captura todos los tipos de cambios que pudieran ocurrir. Por otra parte, ejecutar constantemente dichas funciones provoca un consumo innecesario de recursos del sistema, debido a que este proceso funciona en todo momento aunque no detecte cambios; ocupándose espacio en el disco duro, pues PostgreSQL almacena en los *logs*[2] creados en su carpeta de instalación, la implementación de una función cada vez que se ejecuta.

A causa de lo explicado anteriormente surge la siguiente interrogante: ¿Cómo contribuir a evitar conflictos en la réplica de datos en bases de datos gestionadas en PostgreSQL y reducir el uso de recursos del sistema, utilizando el Replicador de datos Reko? La cual constituye el **problema a resolver**.

De aquí que el **objeto de estudio** sea el proceso de captura de cambios de estructura en bases de datos gestionadas en PostgreSQL y el **campo de acción** el proceso de captura de cambios de estructura en bases de datos gestionadas en PostgreSQL utilizando *triggers DDL*[3].

Para dar solución al problema planteado se define como **objetivo general**: Desarrollar un mecanismo de captura de cambios de estructura mediante *triggers DDL*, en bases de datos gestionadas en PostgreSQL utilizando el Replicador de datos Reko, para contribuir a evitar conflictos en la réplica de datos y reducir el uso de recursos del sistema.

Para lograr el cumplimiento del objetivo, se proponen las siguientes **tareas de investigación**:

[2] Log: Registro de actividad de un sistema, que generalmente se guarda en un fichero de texto.
[3] *Trigger* o disparador: Procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación en una BD.
Trigger DDL: Procedimiento que se ejecuta específicamente cuando se produce una acción sobre la estructura de una BD.
DDL: *Data Definition Language* (Lenguaje de Definición de Datos), abarca las cláusulas CREATE, ALTER y DROP.

1. Realización del marco conceptual para precisar los principales conceptos que se emplean en la investigación.
2. Elaboración del estado del arte de las herramientas de réplica que necesiten dentro de sus procesos la captura de cambios de estructura, para sintetizar los resultados alcanzados en la revisión bibliográfica e indagaciones realizadas relacionadas con el tema.
3. Selección de la metodología para definir los métodos y técnicas necesarias que guiarán el desarrollo.
4. Descripción de las herramientas, tecnologías y lenguajes a utilizar para definir el ambiente de desarrollo.
5. Definición de los requisitos funcionales y no funcionales para identificar las capacidades que tienen que ser alcanzadas por el sistema para cumplir los objetivos trazados.
6. Descripción de la arquitectura que soporta la implementación de las funcionalidades.
7. Realización del análisis y diseño del mecanismo para describir los artefactos.
8. Implementación de las funcionalidades que dan cumplimiento a los requisitos identificados.
9. Realización de pruebas al mecanismo en entornos de producción para valorar la calidad del producto implementado.
10. Valoración de los resultados obtenidos.

Posibles resultados:

1. Debe quedar elaborado un informe detallado con la base teórica-práctica sobre la cual se sustenta la solución propuesta.
2. Debe quedar completado un mecanismo de captura de cambios de estructura en bases de datos gestionadas en PostgreSQL empleando *triggers DDL*, para el Replicador de datos Reko.

Para realizar la investigación se hará uso de los **métodos científicos**:

De nivel teórico:

- **Histórico – Lógico:** Para realizar el estudio acerca de los sistemas o soluciones similares, además de los lenguajes, herramientas y metodologías para el desarrollo del sistema que se propone.
- **Modelación:** Para darle cumplimiento a los requisitos funcionales y no funcionales asociados al sistema.

El presente documento consta de tres capítulos que a su vez se dividen en epígrafes y estos en sub-epígrafes de acuerdo al nivel de detalle que requiere el contenido abordado en cada uno de ellos. A continuación, se explica brevemente el contenido de cada capítulo:

Capítulo I. Fundamentación Teórica: en este capítulo se define la base teórica de la presente investigación, incluye el análisis de la información existente acerca del tema a tratar y las tendencias actuales que existen en el mundo. También incluye una descripción dando un sustento teórico a la selección de las herramientas y *frameworks*[4] a utilizar para resolver el problema planteado.

Capítulo II. Características y Diseño del sistema: en este capítulo se realiza una breve descripción de la solución propuesta, sus requisitos funcionales y no funcionales. Asimismo, se representan los artefactos generados a través del proceso de desarrollo del sistema.

Capítulo III. Implementación y Pruebas: en este capítulo se muestra parte de la implementación de la solución, especificando el estándar de codificación. Contiene además las pruebas que le serán aplicadas a la herramienta y el análisis de los resultados obtenidos en este proceso.

[4] *Framework* (Marco de trabajo): Conjunto de funcionalidades y soluciones ya implementadas que permiten agilizar el trabajo del programador mediante la reutilización de código.

Capítulo I. Fundamentación Teórica

Este capítulo explica los principales conceptos asociados al proceso de captura y réplica de cambios de estructura en bases de datos gestionadas en PostgreSQL. Además, se realiza una breve descripción de las herramientas y metodologías de desarrollo a utilizar para alcanzar los objetivos trazados.

1.1 Marco Conceptual

A continuación, se muestran algunos conceptos tratados durante el transcurso de la investigación, a fin de introducir el tema y lograr una mejor comprensión del trabajo en cuestión.

1.1.1 Base de Datos (BD)

Según Manuel Sierra[5], una BD es un sistema informático a modo de almacén, donde se guardan grandes volúmenes de información. Su principal función es que automatiza el acceso a la misma y permite acceder a ella de manera rápida y fácil, además de realizar cambios de manera más eficiente. [1]

Rosa M. Mato[6] define una BD como un conjunto de datos interrelacionados, almacenados con carácter más o menos permanente en la computadora, o sea, una colección de datos variables en el tiempo. [2]

Adrian Hernández plantea que una BD es conjunto de datos almacenados, variables en el tiempo, relacionados entre sí y utilizados por aplicaciones externas. [3]

Para la investigación se considera que una BD es un sistema informático que puede almacenar un gran volumen de datos interrelacionados, variables en el tiempo y accesibles de manera rápida y fácil por aplicaciones externas, además de que permite realizar cambios de manera eficiente.

1.1.2 Sistema de Bases de Datos Distribuidas (SBDD)

[5] Manuel Sierra: Analista y programador informático con experiencia en la tecnología JAVA, PHP, C#.
[6] Rosa M. Mato: Licenciada en Cibernética Matemática, Profesora Auxiliar en el Centro de Estudios de. Ingeniería de Sistemas (CEIS).

Según el autor Jofman Pérez[7], un sistema de bases de datos distribuidas está compuesto por múltiples sitios de bases de datos ligados por un sistema de comunicaciones, de manera que desde cualquier sitio se puede acceder a los datos en todo el sistema como si estuvieran en un mismo lugar. [4]

Enrique Basto y Luis A. Carbonel[8] definen a un SBDD como un conjunto de bases de datos lógicamente relacionadas, las cuales se ubican de forma distribuida en diferentes sitios interconectados por una red de comunicaciones, teniendo la capacidad de procesamiento autónomo lo cual posibilita la realización de operaciones locales o distribuidas. [5]

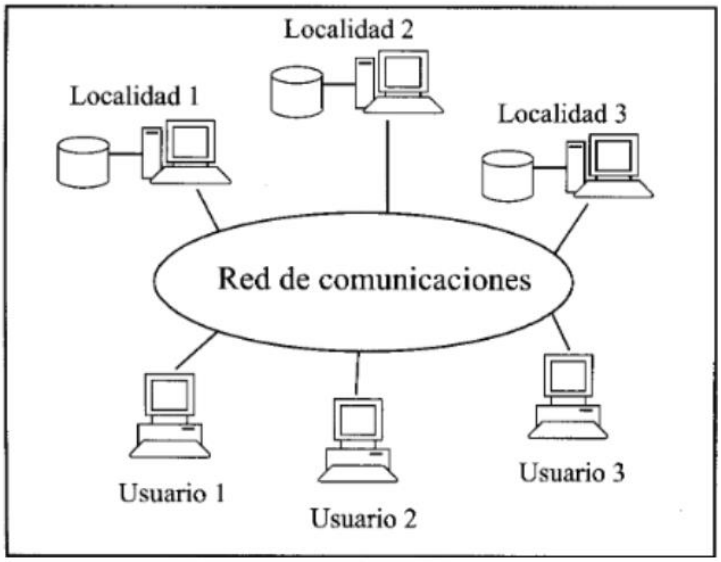

Figura 1: Ejemplo de un SBDD
Fuente: [5]

Para la investigación se considera que los SBDD son un conjunto de bases de datos u otros SBDD interconectados por medio de la red, que almacenan información perteneciente lógicamente a un único sistema, pero están ubicados físicamente en diferentes lugares. Dichos sistemas permiten el acceso a sus datos como si estuvieran ubicados en un mismo lugar.

[7] Jofman Pérez: Ingeniero en Ciencias Informáticas y Master en Informática Aplicada.
[8] Enrique Basto y Luis A. Carbonel: Ingenieros en Ciencias Informáticas.

1.1.3 Replicación de datos

El autor coincide con el concepto planteado por Daniel E. Castro[9] y Ramiro Pérez[10]:

La replicación de datos consiste en el transporte de datos entre dos o más servidores, permitiendo que ciertos datos de la base de datos estén almacenados en más de un sitio, y así aumentar la disponibilidad de los datos y mejorar el rendimiento de las consultas globales. El modelo de replicación está formado por: publicador, distribuidor, suscriptor, publicación, artículo y suscripción; y varios agentes responsabilizados de copiar los datos entre el publicador y el suscriptor. A los tipos básicos de replicación (de instantáneas, transaccional y de mezcla), se le incorporan opciones para ajustarse aún más a los requerimientos del usuario. [6]

1.1.4 Características de las Réplicas de Datos

Propósitos de la replicación: [3]

- **Disponibilidad:** los datos son accesibles desde varios sitios, aun cuando algunos sitios han cerrado.
- **Rendimiento:** la replicación permite localizar los datos más cerca de los puntos de acceso de los usuarios reduciendo los tiempos de respuestas.

La replicación de datos se clasifica teniendo en cuenta tres elementos fundamentales: [3]

1. Ambientes de replicación:
   - **Maestro-esclavo** (*master-slave*): en este caso la replicación se realiza en un sólo sentido: desde un nodo maestro a uno o varios esclavos.
   - **Multi-maestro** (*multi-master*): se configuran varios nodos como maestros, permitiendo que ocurra la replicación en todos los sentidos.
2. Forma de transmitir los cambios en el entorno de replicación:
   - **Sincrónica:** los cambios ejecutados en un nodo maestro son aplicados instantáneamente en el nodo origen y en los nodos destinos dentro de una misma transacción. En caso de no poder ejecutarse la acción en cualquiera de los nodos, la transacción completa es cancelada. Requiere una alta disponibilidad de recursos de red.

[9] Lic. Daniel Eduardo Castro Morell: Licenciado en Cibernética-Matemática. Profesor Instructor del Departamento de Computación. Universidad Central "Marta Abreu" de las Villas. Santa Clara. Cuba.
[10] Dr. Ramiro Pérez Váquez: Doctor en Ciencias Técnicas. Profesor Titular del Departamento de Computación. Universidad Central "Marta Abreu" de las Villas. Santa Clara. Cuba.

- **Asincrónica:** los cambios ejecutados en una tabla son almacenados y enviados posteriormente al resto de los nodos del entorno cada cierto intervalo de tiempo.

3. Forma de capturar y almacenar los cambios a replicar:
   - **Basada en *triggers*:** se crean una serie de *triggers* en las bases de datos que permiten capturar las operaciones de inserción, actualización y eliminación realizadas sobre las tablas a replicar.
   - **Basada en *logs*:** se sostiene en la lectura de *logs* de cambios que proporcionan algunos gestores como Oracle.

La réplica de datos puede ocurrir en dos tipos de entornos diferentes: [3]

- **Entorno homogéneo:** los procesos de réplicas ocurren en los diferentes ambientes que usan el mismo gestor de bases de datos.
- **Entorno heterogéneo:** la réplica heterogénea es aquella que es implementada sobre gestores de bases de datos diferentes, por ejemplo: bases de datos funcionando con MySQL y otra con PostgreSQL.

1.1.5 Conflicto de replicación

El *IBM Knowledge Center* (Centro de Conocimiento de IBM), plantea que un conflicto de replicación se produce cuando varios usuarios editan el mismo documento en réplicas distintas entre una sesión de replicación y otra. [7]

Según Randy Urbano[11], pueden producirse conflictos de replicación en un entorno de replicación que permite actualizaciones simultáneas a los mismos datos en varios sitios. Por ejemplo, cuando dos transacciones procedentes de diferentes sitios actualizan la misma fila casi al mismo tiempo, puede producirse un conflicto. [8]

El autor Helian Rivera[12], plantea que la primera opción al desarrollar un ambiente de replicación debe ser la de diseñarlo de manera que se eviten los conflictos de réplica. Al usar varias técnicas, la mayoría de los sistemas pueden evitar los conflictos en un gran porcentaje de los datos que se replican. Aun así, debido a que se requiere que en estos sistemas existan datos que pueden ser actualizados desde múltiples sitios siempre existe la posibilidad de que surjan conflictos de réplica. [9]

[11] Randy Urbano: Escritor técnico de Oracle.
[12] Helian Rivera: Ingeniero en Ciencias Informáticas.

Los conflictos se pueden separar en 3 tipos: [9]

Conflictos de actualización: Ocurren cuando dos transacciones se originan de sitios diferentes, mientras que una elimina una fila la otra transacción actualiza. Esto sucede porque en este caso no existe la fila para actualizar, ya que fue previamente borrada.

Conflictos de unicidad: Ocurren cuando la replicación de una fila intenta violar la integridad de la entidad, como por ejemplo la llave primaria (PRIMARY_KEY) o restricción única (UNIQUE_CONSTRAINT).

Conflictos de eliminación: Ocurren cuando se trata de eliminar una tulpa que ya no existe.

1.1.6 Lenguaje de Definición de Datos (*DDL*)

Para la presente investigación, el autor coincide con lo expresado por Enrique Basto y Luis A. Carbonel:

El Lenguaje de Definición de Datos se conoce genéricamente como *DDL* o *Data Definition Language*. Define y describe los objetos de la base de datos, su estructura, relaciones y restricciones. En la práctica puede consistir en un subconjunto de instrucciones de otro lenguaje informático. [5]

Posee tres operaciones básicas: [5]

- *Create*: para crear nuevas tablas, campos, índices, *triggers*, funciones, etcétera.
- *Drop*: para eliminar los elementos anteriores.
- *Alter*: para modificar los elementos anteriores.

1.2 Análisis de trabajos relacionados

El análisis de algunos replicadores, permitió recopilar la información y así sintetizar los conocimientos necesarios para proponer una solución correcta. La investigación se centra en las herramientas capaces de realizar réplica de estructura en bases de datos gestionadas en PostgreSQL.

1.2.1 SymmetricDS

SymmetricDS es un software de código abierto escrito en Java bajo la GPL[13] de GNU para la replicación de base de datos. Soporta múltiples suscriptores con una dirección o la replicación de

[13] GPL: *General Public License* (Licencia Pública General)

datos asincrónica bidireccional. Utiliza tecnologías web y bases de datos para replicar tablas entre bases de datos relacionales, casi en tiempo real. El software fue diseñado para escalar a un gran número de bases de datos, trabajar a través de conexiones de bajo ancho de banda, y soportar períodos de interrupción de la red. Además, es capaz de realizar réplicas de datos entre los gestores MySQL, Oracle, SQL Server, PostgreSQL, entre otros. [10]

Replicar el esquema de BD en los nodos remotos y sincronizar los cambios de esquema para las tablas cuando se alteran es posible con SymmetricDS utilizando un par de técnicas simples y mediante *triggers*. En primer lugar, las tablas se pueden crear y rellenar con una carga inicial de datos cuando el nodo se registra. En segundo lugar, las definiciones de tabla se pueden enviar a los nodos necesarios para crear o modificar las tablas usando el *DDL* de la BD. [11]

Existen tres pasos para la sincronización de los cambios de esquema:

- **Modificación de la Base de datos**: se ejecuta el lenguaje de definición de datos *DDL* en la base central de datos. Esto debe hacerse durante un período razonablemente sin contención. crea, elimina y altera las tablas, pero no actualiza ningún dato.
- **Sincronización con *triggers***: se ejecuta el "***sync-triggers***", proceso en SymmetricDS para detectar el cambio de esquema. Se vuelven a crear los factores desencadenantes para que coincidan con las definiciones de la tabla. Una manera fácil de hacer esto es reiniciar el servicio, o ejecutando el siguiente comando.
- **Enviar esquema**: se ejecuta el comando "***send-symadmin***" con esquema de subcomando para enviar los cambios de esquemas a nodos remotos. Este comando detecta qué cambios hacer a la tabla a través de plataformas de BD diferentes. Si conoce el *SQL* (Lenguaje de Consulta Estructurada) específico que debe ejecutar para su BD, puede ejecutar el comando "***send-symadmin***" con el subcomando "***send-sql***" para enviar el cambio *SQL* a nodos remotos. Este subcomando admite el envío de un nodo o grupos de nodos. Los cambios de esquema se ponen en cola, en captura de datos modificados, por lo que se reproducirán en el mismo orden en otros nodos. Este paso es alternativo. [10]

1.2.2 DBReplicator

Es un software de replicación multi-maestro, abierto para la red de aplicaciones Java. [12]

Las principales características de DBReplicator son las siguientes:

- Principales servidores de BD soportados: MySQL, Oracle, PostgreSQL, Microsoft SQL Server.
- Sincronización bidireccional.
- Independiente de la plataforma.
- Detección y resolución de conflictos.
- Tareas programadas.
- Codificación de caracteres que no estén en formato ASCII para cuando se crean archivos de sincronización en XML.
- Creación de tablas en el subscriptor (destino), si la tabla que replica desde el publicador (origen) no está presente en la base de datos del subscriptor. [12]

1.2.3 DBMoto Cloud Edition

DBMoto Cloud Edition es una herramienta tecnológica que ofrece una solución para ejecutar operaciones de captura de datos entre BD sin conexión directa a la red, en lugar de un *web service*[14]. Para un óptimo rendimiento, el producto es diseñado para capturar cambios en la BD origen y propagar esos cambios al o los destinos. Inicialmente, es necesario generar una copia completa de las tablas envueltas y actualizar la BD destino con un completo conjunto de datos capturados. [13]

DBMoto Cloud Edition provee la replicación y transformación de datos en tiempo real. Incorpora tecnología de acceso para garantizar un alto rendimiento y un óptimo nivel de seguridad en la replicación de sus datos, con el valor agregado de estar disponible para transmitir datos relacionales desde el origen al destino utilizando la nube. Captura de datos, el cual reduce significativamente la cantidad de movimientos de datos para actualización y puede ser usada para actualizar datos remotos atrás de *firewalls*[15]. [13]

Sus principales características son:

- Soporta gestores de BD como: *Oracle*, *Microsoft SQL Server*, *MySQL*, *Server Enterprise*, *PostgreSQL*, y *MS Access*.

[14] W*eb service*: Servicio web, tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.
[15] *Firewalls*: Sistema de defensa basado en el hecho de que todo el tráfico de entrada o salida a la red debe pasar obligatoriamente por un sistema de seguridad capaz de autorizar, denegar, y tomar nota de todo aquello que ocurre, de acuerdo con una política de control de acceso entre redes.

- Modos de replicación y captura de datos: *Refresh* y *Mirroring*[16].
- Replicación de datos desde origen al destino, usando la nube y *web service* para transferencia de datos.
- Soporte para múltiples BD (origen y destino).
- No requiere programación en las plataformas de BD de origen y destino.
- Habilidad para leer *logs* de transacciones de BD (para hacer más eficiente el acceso a los datos).
- Completo *log* de reportes y accesibilidad.
- Poderosa herramienta visual con información sobre el estatus de la replicación.
- Creación automática de tablas de destino. [13]

1.2.4 Resultados

A partir del estudio realizado, se concluye que las herramientas existentes no satisfacen los requerimientos necesarios para solucionar la problemática, debido a que:

- En ciertos casos se hace referencia a la realización de los procesos, sin ofrecer detalles de su funcionamiento interno, ni de cómo se realizan.
- Sólo se posee el código fuente de SymmetricDS.
- La integración con Reko de cualquiera de las herramientas analizadas que replican estructura, reduciría considerablemente el rendimiento del sistema, ya que esta funcionalidad no puede ser aislada en ninguno de los casos.
- SymmetricDS es la herramienta que mejor cumple con los requerimientos deseados, pero el inicio del proceso de réplica de estructura y las configuraciones se realizan mediante comandos en una consola de administración, lo que implica que los usuarios deben poseer un conocimiento avanzado sobre el tema.

Esta situación condujo a la decisión de brindar a Reko nuevas funcionalidades para cumplir con el objetivo; ya que, al ser desarrollado en la Universidad de Ciencias Informáticas, se evita el pago de licencias propietarias y se contribuye a la independencia tecnológica de Cuba.

1.3 Metodología de Desarrollo de Software

Con el fin de hacer el comportamiento de un software más predecible y eficiente, las metodologías proponen un proceso de desarrollo exigente, cuyo objetivo principal es aumentar la calidad del

[16] *Refresh* y *Mirroring*: Actualizar y reflejar.

software en todas sus etapas de producción, haciendo énfasis en torno a la idea de producir con la mayor calidad en el menor tiempo. [14]

La Universidad de las Ciencias Informáticas, en su proceso de madurez del desarrollo de software, define como política a seguir en los proyectos productivos la utilización de la metodología Proceso Unificado Ágil (AUP) adaptada al ciclo de vida definido para la actividad productiva de la universidad, por lo que se decide su utilización para guiar el proceso de desarrollo de la solución. [14]

1.3.1 Descripción de la metodología AUP

El Proceso Unificado Ágil fue creado por Scott Ambler, es una versión simplificada del Proceso Unificado de Rational (RUP). Este describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. Se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y por ser iterativo e incremental. Algunas técnicas usadas por AUP incluyen el desarrollo orientado a pruebas, modelado y gestión de cambios ágiles, y refactorización de base de datos para mejorar la productividad. [14]

1.3.2 Fases de AUP

Al igual que en RUP, en AUP se establecen cuatro fases que transcurren de manera consecutiva: Inicio, Elaboración, Construcción y Transición. De las 4 fases que propone AUP se decide para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando el objetivo de la misma, se unifican las restantes 3 fases de AUP en una sola, llamada Ejecución y se agrega la fase de Cierre. Para una mayor comprensión se muestra la siguiente tabla: [14]

Tabla 1: Fases de AUP

Fases AUP	**Fases Variación AUP-UCI**	**Objetivos de las fases (Variación AUP-UCI)**
Inicio	Inicio	Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.

Elaboración Construcción Transición	Ejecución	En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.
	Cierre	En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Fuente: [14]

1.3.3 Disciplinas de AUP

AUP define 7 disciplinas (4 ingenieriles y 3 de gestión de proyectos): Modelo, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyectos y Entorno. Para el ciclo de vida de los proyectos de la UCI se definen 7 disciplinas también, pero a un nivel más atómico. Los flujos de trabajos: Modelado de negocio, Requisitos y Análisis y diseño en AUP están unidos en la disciplina Modelo, en la variación para la UCI se consideran a cada uno de ellos disciplinas. Se mantiene la disciplina Implementación. En el caso de Prueba se desagrega en 3 disciplinas: Pruebas Internas, de Liberación y Aceptación. Las restantes 3 disciplinas de AUP asociadas a la parte de gestión para la variación UCI se cubren con las áreas de procesos que define CMMI DEV v1.3 para el nivel 2. Para una mayor comprensión se muestra la siguiente tabla: [14]

Tabla 2: Disciplinas de AUP – UCI

Disciplinas AUP	**Disciplinas Variación AUP-UCI**	**Objetivos (Variación AUP-UCI)**
Modelo	Modelado de negocio (Opcional)	Destinada a comprender los procesos de una organización y cómo funciona el negocio que se desea informatizar para tener garantías de que el software desarrollado va a cumplir su propósito.
	Requisitos	Enfocada a desarrollar el modelo del sistema que se va a construir. Esta disciplina comprende la administración y gestión de los requisitos funcionales y no funcionales del producto.

	Análisis y diseño	En esta disciplina los requisitos pueden ser refinados y estructurados para conseguir una comprensión más precisa de estos, y una descripción que sea fácil de mantener y ayude a la estructuración del sistema. Además, se modela el sistema y su forma para que soporte todos los requisitos, incluyendo los requisitos no funcionales.
Implementación	Implementación	En la implementación, a partir de los resultados del Análisis y Diseño, se construye el sistema.
Prueba	Pruebas interna	Se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas.
	Pruebas de liberación	Pruebas diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación.
	Pruebas de Aceptación	Es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido.
Gestión de configuración	Se cubren con las áreas de procesos Planeación de Proyecto, Monitoreo y Control de Proyecto, y Administración de la Configuración que propone CMMI DEV v1.3. Las mismas son áreas de procesos de gestión y soporte respectivamente.	
Gestión de proyecto		
Entorno		

Fuente: [14]

1.3.4 Escenarios para la disciplina de requisitos

Existen tres formas de encapsular los requisitos: Casos de Uso del Sistema (CUS), Historias de Usuario (HU) y Descripción de requisitos por proceso (DRP), agrupados en cuatro escenarios condicionados por el Modelado de negocio. [14]

Para este escenario en específico no se modela el negocio, sólo se puede modelar el sistema a partir de las HU. Es aplicado a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. Además, el cliente siempre acompañará al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no muy extensos, pues una HU no debe poseer demasiada información. [14]

1.4 Tecnologías, Lenguajes y Herramientas

El software Replicador de Datos Reko desde sus inicios ha utilizado herramientas y lenguajes de código abierto para su implementación. Siguiendo este principio y con el objetivo de mantener un estándar en el desarrollo se decide utilizar las mismas herramientas y lenguajes en las que fue desarrollado. A continuación, se hace una breve descripción para un mejor entendimiento.

Visual Paradigm para UML 8.0: herramienta multiplataforma diseñada para soportar el ciclo de vida completo del proceso de desarrollo del software, a través de la representación de todo tipo de diagramas. Con capacidad de ingeniería directa e inversa enfocada al negocio que generan un software de mayor calidad. [15]

JEE 1.7.71: *Java Enterprise Edition* es una plataforma que encierra una colección de especificaciones que definen una infraestructura para desarrollar aplicaciones distribuidas multicapa que facilita el desarrollo de aplicaciones distribuidas en Java, ofrece un marco y una serie de convenciones, junto un conjunto de servicios sobre los cuales desarrollar aplicaciones multicapa. [16]

Java: es un lenguaje de uso general, orientado a objeto, concurrente y uno de los más populares a nivel mundial, particularmente para el desarrollo de aplicaciones cliente-servidor en la web. Está diseñado para tener tan pocas dependencias de implementación como se pueda y su intención es permitir que una vez escrito el programa por los desarrolladores se pueda ejecutar en cualquier plataforma. [17]

UML 2.0: es un lenguaje de modelado estándar para el desarrollo de sistemas de software orientado a partir del paradigma de Programación Orientada a Objetos, permitiendo especificar, visualizar, construir y documentar todos los elementos que conforman dichos sistemas. [18]

Eclipse STS 3.6.1: *Spring Tool Suite* (STS) es una herramienta libre desarrollada por *Spring Source*, basada actualmente en Eclipse 3.x, para construir aplicaciones empresariales

enriquecidas por los proyectos de Spring Source. Una de las principales razones para emplear STS es que incluye herramientas para el desarrollo en lenguaje Java, para Spring. [19]

JDBC[17] **4.0:** es un componente de software que permite a las aplicaciones escritas en Java interactuar con una base de datos a través del dialecto SQL sin importar el proveedor de base de datos que se utilice ni el sistema operativo donde se ejecute. [20]

PostgreSQL 9.5: es un sistema de gestión de bases de datos objeto relacional, distribuido bajo licencia BSD14 y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado. [21]

Conclusiones Parciales

A partir del análisis e investigación de los aspectos abordados en este capítulo se puede concluir que:

1. Las herramientas existentes que replican cambios de estructura en BD gestionadas en PostgreSQL, no pueden ser debidamente integradas a Reko o no ofrecen detalles de cómo implementar el proceso de captura de estos cambios, por lo que se decide brindar a Reko nuevas funcionalidades para cumplir con el objetivo.
2. Para la realización de la solución se seleccionó la metodología de desarrollo de software AUP-UCI, pues es recomendada por la Universidad de Ciencias Informáticas para el desarrollo de proyectos productivos y es parte del marco de trabajo actual del equipo de desarrollo de Reko.
3. Debido a que la solución se va a desarrollar sobre el replicador de datos Reko, se decidió utilizar las mismas tecnologías, estándares, lenguajes y herramientas que actualmente tiene definido este proyecto, para mantener un estándar de trabajo.

[17] JDBC: Java Data Base Conectivity (Conector de Java a Base de Datos)

Capítulo II. Características y Diseño del Sistema

En este capítulo se describirá la propuesta de solución mediante el análisis y diseño del sistema, tomando como referencia los conceptos y términos definidos en el primer capítulo, y regido por la metodología de desarrollo AUP – UCI.

2.1 Descripción del Proceso a Automatizar

Mantener actualizados los nodos de un SBDD, es la tarea fundamental para un replicador de datos, de esta forma se garantiza la disponibilidad y el acceso a la información como si se tratase de un único nodo.

El Replicador de datos Reko funciona en un entorno de replicación multi-maestro con una instancia en cada nodo. Los nodos pueden estar agrupados en etiquetas. Desde cada nodo puede configurarse la réplica hacia otro nodo o hacia una etiqueta. A partir de un nodo origen o fuente pueden realizarse distintas configuraciones según los nodos y/o etiquetas destino que se definan. [22]

Actualmente, Reko posee un mecanismo de captura de cambios de estructura utilizando funciones creadas en PostgreSQL, que se ejecutan constantemente a partir de tareas programadas desde Reko. Al ejecutar estas funciones, se revisa casi toda la estructura de la BD en el nodo origen, comparándola con las tablas de captura en busca de cambios, para luego registrarlos en una tabla de control.

Partiendo de lo planteado anteriormente, se propone desarrollar un nuevo mecanismo de captura de cambios de estructura empleando *triggers DDL*. Debido a que los mismos sólo se disparan y ejecutan un procedimiento, en el momento en que suceden las acciones para las cuales están destinados, sólo se capturarán los cambios de estructura en el momento justo en que ocurran; evitando así tener que comprobar la estructura de la BD constantemente y reduciendo el consumo de recursos del sistema.

Por otra parte, PostgreSQL a partir de su versión 9.5, ofrece nuevas características en sus *triggers DDL* que permiten obtener información sobre la ocurrencia de casi todos los tipos de cambios de estructura en las BD que gestiona. Empleando estas nuevas prestaciones, el nuevo

mecanismo permitirá a Reko incorporar la capacidad captura de algunos cambios que actualmente no reconoce, lo que contribuirá a evitar los conflictos que surgen en la réplica de datos cuando la estructura de las BD origen y destino difieren.

2.2 Modelo del Sistema

Un modelo describe lo que debe hacer un sistema, pero no cómo implementarlo. Idealmente, una representación de un sistema, debería mantener toda la información sobre la entidad que se está representando. [23]

2.2.1 Requisitos Funcionales (RF)

Los requisitos funcionales son declaraciones de las funcionalidades que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones específicas. Estos requisitos dependen del tipo de software que se desarrolle, de los posibles usuarios del software y del enfoque general tomado por la organización al redactar requisitos. [24]

A continuación, se muestran los requisitos funcionales definidos para la solución propuesta:

RF1 Crear tablas de control de cambios de estructura
RF2 Capturar estructura inicial
RF3 Crear *trigger* de captura de cambios de tipo CREATE / ALTER
RF4 Crear *trigger* de captura de cambios de tipo DROP
RF5 Eliminar *triggers*

2.2.2 Especificación de Requisitos Funcionales

Tabla 3: Especificación de requisitos funcionales

No.	Nombre	Descripción	Prioridad	Complejidad
RF1	Crear tablas de control de cambios de estructura	Permite crear las tablas donde se registrarán los cambios de estructura ocurridos en la base de datos origen, según la configuración de réplica establecida por el usuario.	Alta	Baja
RF2	Capturar estructura inicial	Permite crear y ejecutar en PostgreSQL la función que registrará en las tablas de control la estructura actual de la BD.	Alta	Media
RF3	Crear *trigger* de captura de cambios de tipo CREATE / ALTER	Permite crear el *trigger* que se disparará ante un cambio estructural de tipo CREATE o ALTER y su respectiva	Alta	Alta

		función en PostgreSQL, según la configuración de réplica establecida.		
RF4	Crear *trigger* de captura de cambios de tipo DROP	Permite crear el *trigger* que se disparará ante un cambio estructural de tipo ALTER y su respectiva función en PostgreSQL, según la configuración de réplica establecida.	Alta	Media
RF5	Eliminar *triggers*	Permite eliminar los *triggers* existentes y sus respectivas funciones para detener el proceso de captura de cambios de estructura o para restablecer el mecanismo de captura según la configuración establecida.	Alta	Baja

Fuente: elaboración propia

2.2.3 Requisitos no Funcionales (RNF)

Los requerimientos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. Los requisitos no funcionales a menudo se aplican al sistema en su totalidad. Normalmente apenas se aplican a características o servicios individuales del sistema. [24]

Para la realización de la solución propuesta, se pretende respetar los requisitos no funcionales definidos para Reko por su equipo de desarrollo, a continuación, se describen los que tiene una mayor incidencia sobre la solución propuesta:

Tabla 4: Requisitos No Funcionales

Requisito no Funcional	**Descripción**	**Sub-atributos de calidad**
Usabilidad	Capacidad del producto que permite al usuario operarlo y controlarlo con facilidad y de proteger a los usuarios de cometer errores.	• Operabilidad • Protección contra errores de usuarios
Fiabilidad	Capacidad del producto para operar según lo previsto en presencia de fallos de hardware o software, así como de recuperar los datos directamente afectados y reestablecer el estado deseado del sistema en caso de interrupción o fallos; además satisfacer las necesidades de fiabilidad en condiciones normales y de estar operativo y accesible para su uso cuando se requiere.	• Tolerancia a fallos • Recuperabilidad • Madurez • Disponibilidad
Mantenibilidad	Facilidad con la que se puede evaluar el impacto de un determinando cambio sobre el resto del software, diagnosticar las deficiencias o causas de fallos en el software, o identificar las partes a modificar; así como de permitir que sea modificado de forma efectiva y eficiente sin introducir defectos o degradar el	• Analizabilidad • Modificabilidad • Capacidad para ser probado

	desempeño. Además de permitir establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios.	
Adecuación funcional	Grado en el que el conjunto de funciones cubre todas las tareas y objetivos del usuario especificados y proporciona los resultados correctos con el grado necesario de precisión.	• Integridad funcional • Corrección funcional
Eficiencia en el rendimiento	Grado en que los tiempos de respuesta, procesamiento y las tasas de rendimiento de un producto o sistema, así como las cantidades y tipos de recursos utilizados por un producto o sistema, al realizar sus funciones cumplen con los requisitos.	• Comportamiento en el tiempo • Utilización de recursos

Fuente: [3]

2.2.4 Descripción de Historias de Usuario (HU)

Entre los artefactos que define la metodología seleccionada se encuentran las HU que son utilizadas para especificar las funcionalidades que brindará el sistema. Cada HU es una representación de un requerimiento de software escrito en una o dos frases utilizando el lenguaje común del usuario. Representan una forma rápida de administrar los requerimientos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos.

Los requisitos funcionales fueron encapsulados en las siguientes HU:

Tabla 5: HU del RF1

Número: HU 1	**Nombre del requisito:** Crear tablas de control de cambios de estructura
Programador: Liosvel Medina Molina	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 1 día
Riesgo en Desarrollo: • Problemas de fallos eléctricos. • Planificación incorrecta. • Actividades extra-producción.	**Tiempo Real:** 8 horas

Descripción:
Permite crear las tablas donde se registrarán los cambios de estructura ocurridos en la base de datos origen, según la configuración de réplica establecida por el usuario:

La tabla **`reko_structure_control_table`** debe contener registrados todos los cambios de estructura ocurridos en la base de datos origen.

Campo	Tipo	Descripción
id	integer	Identificar cada cambio del resto

schema_name	text	Nombre del esquema donde ha ocurrido el cambio
table_name	text	Nombre de la tabla donde ha ocurrido el cambio
column_name	text	Nombre de la columna donde ha ocurrido el cambio
other_especification	text	Especificación del cambio, ej. nuevo nombre
action_type	text	Tipo de cambio ocurrido, ej. create_schema, add_colum
targets	text	Nodo o conjunto de nodos destino
replic_user	text	Usuario de PostgreSQL que Reko utiliza para acceder a la BD

La tabla `reko_structure_capture_schema` debe contener información del estado actual de cada esquema de la BD.

Campo	**Tipo**	**Descripción**
id_schema	integer	Identificador del esquema en la BD
schema_name	text	Nombre del esquema en la BD
is_updated	boolean	Toma valor `false` cuando el esquema sufre algún cambio, toma valor `true` cuando el cambio se registra en la tabla `reko_structure_control_table`
is_new	boolean	Toma valor `true` justo cuando el esquema es creado, toma valor `false` cuando el cambio se registra en la tabla `reko_structure_control_table`

La tabla `reko_structure_capture_table` debe contener información del estado actual de cada tabla de la BD.

Campo	**Tipo**	**Descripción**
id_table	integer	Identificador de la tabla en la BD
table_name	text	Nombre de la tabla en la BD
is_altered	boolean	Toma valor `true` cuando la tabla sufre algún cambio, toma valor `false` cuando el cambio se registra en la tabla `reko_structure_control_table`
is_new	boolean	Toma valor `true` justo cuando la tabla es creada, toma valor `false` cuando el cambio se registra en la tabla `reko_structure_control_table`
id_schema	integer	Identificador del esquema al cual pertenece la tabla

La tabla `reko_structure_capture_column` debe contener información del estado actual de cada columna de la BD.

Campo	**Tipo**	**Descripción**
id_column	text	Identificador de la columna dentro de la tabla a la cual pertenece
column_name	text	Nombre de la columna
type	text	Tipo de dato de la columna
default_value	text	Valor por defecto de la columna
is_not_null	boolean	Toma valor `true` si la columna no puede tomar valor nulo.
statictics_value	integer	Toma el valor estático de la columna
is_altered	boolean	Toma valor `true` cuando la columna sufre algún cambio, toma valor `false` cuando el cambio se registra en la tabla `reko_structure_control_table`

is_new	boolean	Toma valor `true` justo cuando la columna es creada, toma valor `false` cuando el cambio se registra en la tabla `reko_structure_control_table`
id_table	integer	Identificador de la tabla a la cual pertenece la columna

La tabla `reko_structure_capture_constraint` debe contener información del estado actual de cada restricción de la BD.

Campo	**Tipo**	**Descripción**
id_constraint	integer	Identificador de la restricción en la BD
constraint_name	text	Nombre de la restricción
type	text	Tipo de restricción
expression	character varying	Si es una restricción de tipo `check`, toma el valor de la expresión que acompaña a la restricción
id_table_ref	integer	Si es llave foránea, toma el valor del identificador de la tabla origen
id_column_ref	integer	Si es llave foránea, toma el valor del identificador de la columna de la tabla origen
is_new	boolean	Toma valor `true` justo cuando la restricción es creada, toma valor `false` cuando el cambio se registra en la tabla `reko_structure_control_table`
id_table	integer	Identificador de la tabla a la cual pertenece la restricción
id_column	text	Identificador de la columna a la cual pertenece la restricción

Observaciones:

- Las tablas sólo se crearán si existe alguna configuración de réplica de estructura establecida y si la opción *Replicar cambios de esquemas* está activada

Prototipo elemental de interfaz gráfica de usuario: NA

Fuente: elaboración propia

Tabla 6: HU del RF2

Número: HU 2	**Nombre del requisito:** Capturar estructura inicial
Programador: Liosvel Medina Molina	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 5 días
Riesgo en Desarrollo: • Problemas de fallos eléctricos. • Planificación incorrecta. • Actividades extra-producción.	**Tiempo Real:** 8 días

Descripción:

Permite crear, ejecutar y luego eliminar en PostgreSQL la función `reko_structure_capture_function()` que registrará en las tablas de control la estructura actual de la BD.

Observaciones:

- La función `reko_structure_capture_function()` debe ser creada, ejecutada y eliminada al inicio del proceso de captura y cada vez que cambie la configuración de réplica desde Reko.

Prototipo elemental de interfaz gráfica de usuario: NA

Tabla 7: HU del RF3

Número: HU 3	**Nombre del requisito:** Crear *trigger* de captura de cambios de tipo CREATE / ALTER
Programador: Liosvel Medina Molina	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 15 días
Riesgo en Desarrollo: • Problemas de fallos eléctricos. • Planificación incorrecta. • Actividades extra-producción.	**Tiempo Real:** 21 días
Descripción: Permite crear el *trigger* `reko_on_structure_change_trigger` que se dispara ante cambios estructurales de tipo CREATE o ALTER y su respectiva función `reko_on_structure_change_triggered_function()` en PostgreSQL, al establecer la configuración de réplica.	
Observaciones: • El *trigger* `reko_on_structure_change_trigger` se crea si se establece la opción de Creación o Modificación en la configuración. • La función `reko_on_structure_change_triggered_function()` se crea según las opciones de réplica (Creación, Modificación) establecidas en la configuración.	
Prototipo elemental de interfaz gráfica de usuario: NA	

Fuente: elaboración propia

Tabla 8: HU del RF4

Número: HU 4	**Nombre del requisito:** Crear *trigger* de captura de cambios de tipo DROP
Programador: Liosvel Medina Molina	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 10 día
Riesgo en Desarrollo: • Problemas de fallos eléctricos. • Planificación incorrecta. • Actividades extra-producción.	**Tiempo Real:** 9 días
Descripción: Permite crear el *trigger* `reko_on_structure_drop_trigger` que se disparará ante cambios estructurales de tipo DROP y su respectiva función `reko_on_structure_drop_triggered_function()` en PostgreSQL, al establecer la configuración de réplica.	
Observaciones:	

• El *trigger* `reko_on_structure_drop_trigger` y la función `reko_on_structure_drop_triggered_function()` se crean si se establece la opción de Eliminación en la configuración.
Prototipo elemental de interfaz gráfica de usuario: NA

Fuente: elaboración propia

Tabla 9: HU del RF5

Número: HU 5	**Nombre del requisito:** Eliminar *triggers*
Programador: Liosvel Medina Molina	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 3 horas
Riesgo en Desarrollo: • Problemas de fallos eléctricos. • Planificación incorrecta. • Actividades extra-producción.	**Tiempo Real:** 2 horas
Descripción: Permite eliminar los *triggers* `reko_on_structure_change_trigger` y `reko_on_structure_drop_trigger` y respectivas funciones `reko_on_structure_change_triggered_function()` y `reko_on_structure_drop_triggered_function()`, para detener el proceso de captura de cambios de estructura o para restablecer el mecanismo de captura según la configuración establecida.	
Observaciones: NA	
Prototipo elemental de interfaz gráfica de usuario: NA	

Fuente: elaboración propia

2.3 Descripción de la Arquitectura

Las técnicas metodológicas desarrolladas con el fin de facilitar la programación se engloban dentro de la llamada Arquitectura de Software o Arquitectura lógica. Se refiere a un grupo de abstracciones y patrones que brindan un esquema de referencia útil para guiarnos en el desarrollo de software dentro de un sistema informático. Así, los programadores, diseñadores, ingenieros y analistas pueden trabajar bajo una línea común que les posibilite la compatibilidad necesaria para lograr el objetivo deseado. [9]

Según el autor Carlos Billy Reynoso: la Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones. [25]

La IEEE 1471 -2000 define a la arquitectura de software como: “La organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente, y los principios que orientan su diseño y evolución.” [26]

Lo expuesto anteriormente, permite concluir que la conformación de la Arquitectura durante el desarrollo de un software ofrece una vista con sus principales componentes y las relaciones entre los mismos, facilitando la realización de tareas de implementación simultáneas y la detección temprana de errores.

2.3.1 Estilo Arquitectónico

Microsoft Developer Network define un estilo arquitectónico como: “una lista de tipos de componentes que describen los patrones o las interacciones a través de ellos. Un estilo afecta a toda la arquitectura de software y puede combinarse en la propuesta de solución”. [27]

Según Dresky Ortiz y Carlos J. Bullón, un estilo arquitectónico define una familia de sistemas de software en términos de su organización estructural. Representa los componentes y las relaciones entre ellos con las restricciones de su aplicación y las asociaciones y reglas de diseño para su construcción. [28]

Una arquitectura basada en componentes es un acercamiento basado en la reutilización para definir e implementar componentes débilmente acoplados en sistemas. Esta arquitectura se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas. Esto provee un nivel de abstracción mayor que los principios de orientación por objetos y no se enfoca en asuntos específicos de los objetos como los protocolos de comunicación y la forma como se comparte el estado. [5]

El replicador de datos Reko define su arquitectura basada en componentes, debido a que sus partes encapsulan un conjunto de comportamientos que pueden ser reemplazados por otros. Sus principales componentes son:

Capturador de Cambios: Encargado de capturar los cambios que se realizan en la base de datos y entregarlos al distribuidor.

Distribuidor: Determina el destino de cada cambio realizado en la base de datos, los envía y se responsabiliza de su llegada.

Aplicador: Ejecuta en la base de datos los cambios que son enviados hacia él desde otro nodo de réplica.

Administrador: Permite realizar las configuraciones principales del software como el registro de nodos, configuración de las tablas a replicar y el monitoreo del funcionamiento.

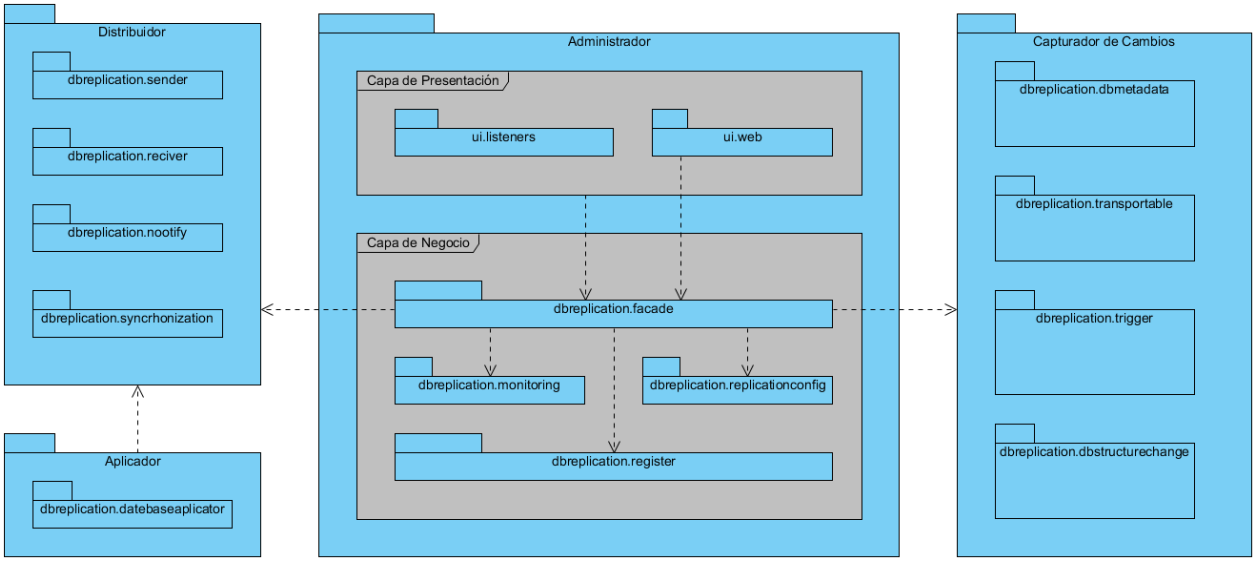

Figura 2: Arquitectura de Reko
Fuente: elaboración propia

A pesar de que la arquitectura es basada en componentes se puede apreciar que el componente **Administración** responde a un modelo multicapas, donde cada capa encapsula un comportamiento y objetivos precisos, permitiendo así que su implementación sea desacoplada con respecto a otra capa y el intercambio de información se realice a través de interfaces. Además de estar separadas lógica y estructuralmente, las capas se encuentran separadas de manera física. [5]

La capa de **presentación** proporciona al usuario una interfaz gráfica con la que puede introducir información al sistema y recibir las notificaciones del mismo. Esta capa sólo se comunica con la capa de **negocio**, donde se reciben las peticiones del usuario y se procesan. También es conocida como capa de lógica de negocio debido a que aquí es donde ese establece las reglas que deben cumplirse. [5]

Para el desarrollo de la solución, se realizarán las principales modificaciones sobre los paquetes `dbreplication.dbstructurechange` y `dbreplication.triggers` del Capturador de Cambios; además de aprovechar algunas funcionalidades ya implementadas de `dbreplication.dbmetadata` y `dbreplication.utils`.

2.3.2 Patrones de diseño

Una buena práctica en la definición de la arquitectura del sistema es el uso de patrones, los cuales brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. [24]

Patrones GRASP[18]

Consisten en un conjunto de guías para la asignación de responsabilidades a clases y objetos en el diseño orientado a objetos. [3]

Experto: Es el principio básico de asignación de responsabilidades, define que 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. Este patrón se aplica a la clase `DBStructureChangeManager`, pues cuenta con toda la información necesaria para iniciar el proceso de captura de cambios de estructura en la BD.

Controlador: Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal manera que es el que recibe todas las peticiones y datos del usuario y ejecuta en las diferentes clases el método solicitado. Este patrón se aplica a las clases `MetadataManager`, `ConfigurationManager` y `DBStructureChangeManager`, debido a que reciben peticiones de la interfaz y responden realizando determinados procedimientos.

Polimorfismo: la responsabilidad de definir la variación de comportamiento basado en tipos se asigna a aquellos para los cuales esta variación ocurre. Esto se logra mediante el uso de operaciones polimórficas. Este patrón se aplica en la clase interfaz `HibernateDialect`, la cual es implementada por `PostgresDialect`, `MySqlServerDialect`, `Oracle9iDialect` y `SqlServerDialect`; cuya función principal es crear las consultas SQL correspondiente a cada tipo de gestor de BD.

Alta cohesión: Define que la información que almacena una clase debe de ser coherente y estar en mayor medida relacionada con la clase. Los elementos con baja cohesión son más difíciles de comprender, reutilizar, mantener y cambiar. Este patrón se aplica a las clases `PostgresDialect` y `DBStructureChangeManager`, pues la información que manejan está completamente relacionada con las funcionalidades que realizan.

Bajo Acoplamiento: Tiene como objetivo que las clases del sistema tengan la menor dependencia entre ellas. De tal manera que, en caso de producirse alguna modificación, el

[18] Patrones Generales de Software de Asignación de Responsabilidades, del inglés General Responsibility Assignment Software Patterns.

impacto sea el mínimo posible en el resto de clases, potenciando la reutilización. Este patrón se aplica en las clases `DBStructureChangeManager`, `DialectUtils`, `HibernateDialect` y `PostgresDialect`, ya que la implementación de la solución se basa en funcionalidades de `DBStructureChangeManager` que dependen secuencialmente del resto, como se muestra en la figura 3:

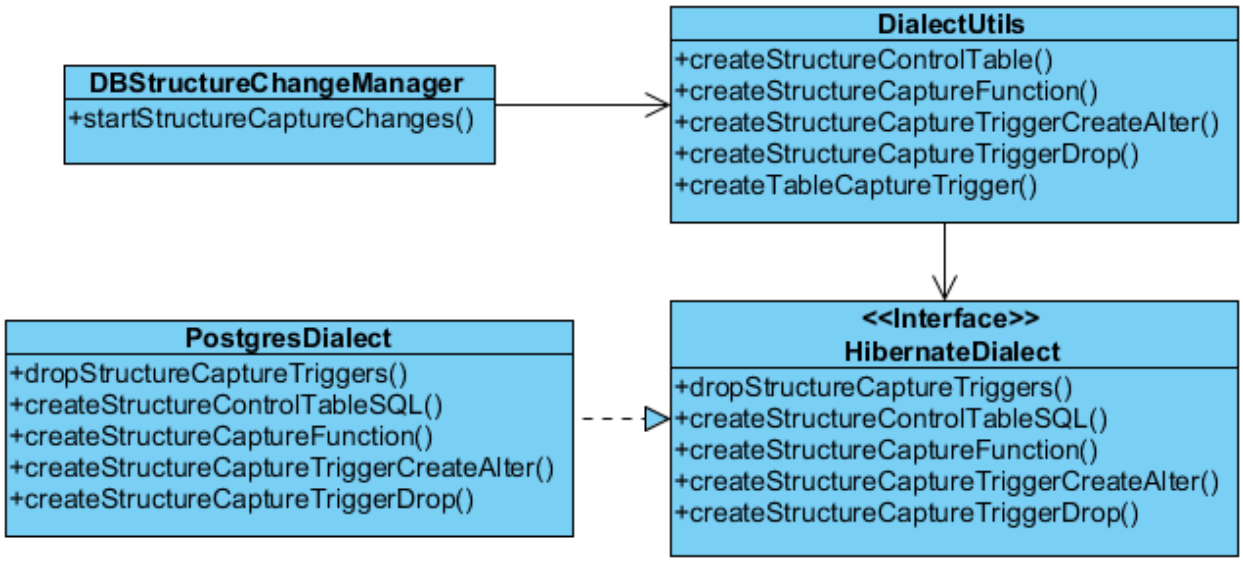

Figura 3: Patrón de Bajo Acoplamiento aplicado a la solución
Fuente: elaboración propia

Patrones GOF[19]

Patrón Fachada: Proporciona una interfaz unificada de alto nivel que, representando a todo un subsistema y facilita su uso. La “fachada” satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas. Por ejemplo, el sistema replicador de datos REKO cuenta con la clase `ReplicationFacade` perteneciente al componente `facade` que ofrece un punto de acceso para iniciar el proceso de captura de cambios de estructura.

2.4 Modelo del Diseño

Un modelo de diseño, a diferencia del modelo de dominio[20], se encuentra más cerca de la solución que se desea obtener y adquiere algunas de las entidades reflejadas en el modelo de dominio para convertirlas en clases. Su propósito es especificar una solución que trabaje y pueda

[19] Banda de los Cuatro, del inglés *Gang-Of-Four*. Nombre con el que se conoce comúnmente a los autores del libro.
[20] El modelo del dominio es una representación de los conceptos u objetos del mundo real, significativos para un problema. Tiene como objetivo fundamental la descripción de las clases más importantes en el sistema y representa conceptos del mundo real, no de los componentes de software.

convertirse fácilmente en código fuente. Representa la solución a las HU definidas y es utilizado como entrada en las tareas de implementación.

2.4.1 Diagramas de Paquetes (DP)

Un DP, es un diagrama UML que permite organizar los elementos del modelo de un sistema, dividiéndolo en partes más pequeñas. Muestra cómo el mismo está dividido en agrupaciones lógicas, mostrando las dependencias entre estas, las cuales pueden indicar el orden de desarrollo requerido. Además, representa una visión general de la arquitectura suministrando una descomposición lógica del sistema. [29]

A continuación, se muestra el diagrama de paquetes definido para el modelo del diseño; el cual se aplica a todas las HU, debido a que para la implementación de cada una se trabajará sobre los mismos componentes:

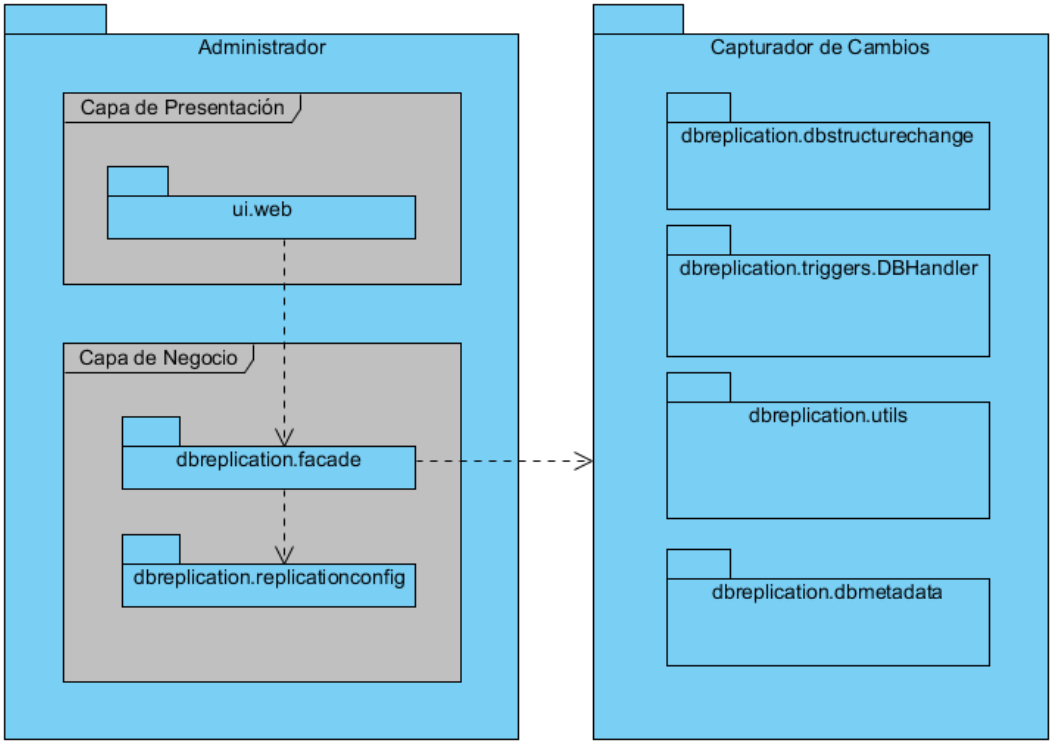

Figura 4: Diagrama de Paquetes de la solución
Fuente: elaboración propia

2.4.2 Diagramas de Clases (DC)

Un DC es una representación de las clases que involucran al sistema y las relaciones entre ellas que pueden ser de asociación, de herencia, de uso y de agregación. Una clase es una definición

de un conjunto de entidades u objetos que comparten los mismos atributos, operaciones y relaciones. [24]

A continuación, se muestran algunos diagramas de clases definidos para el modelo del diseño, para ver los restantes ver Anexos (1, 2):

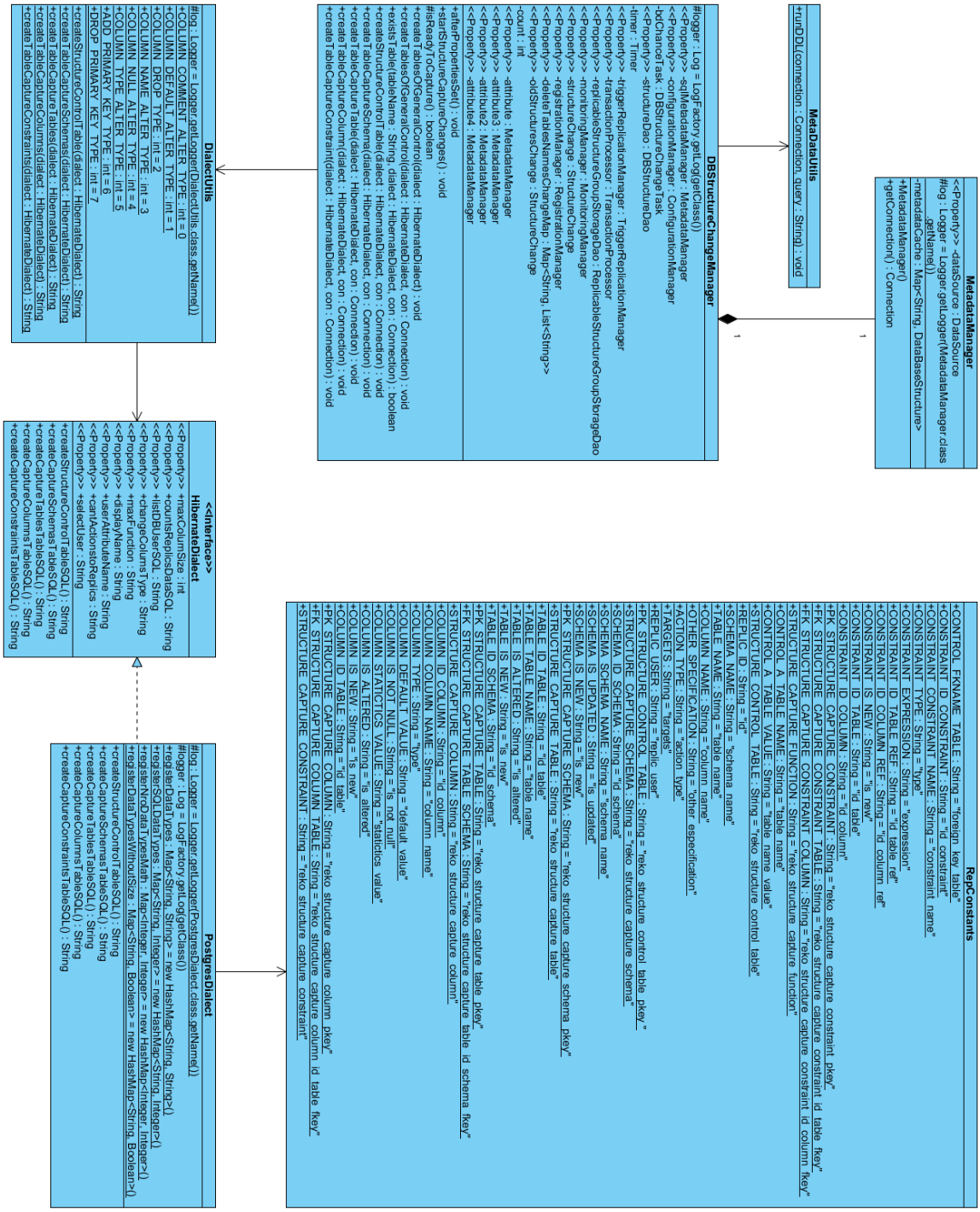

Figura 5: DC Crear tablas de control de cambios de estructura
Fuente: elaboración propia

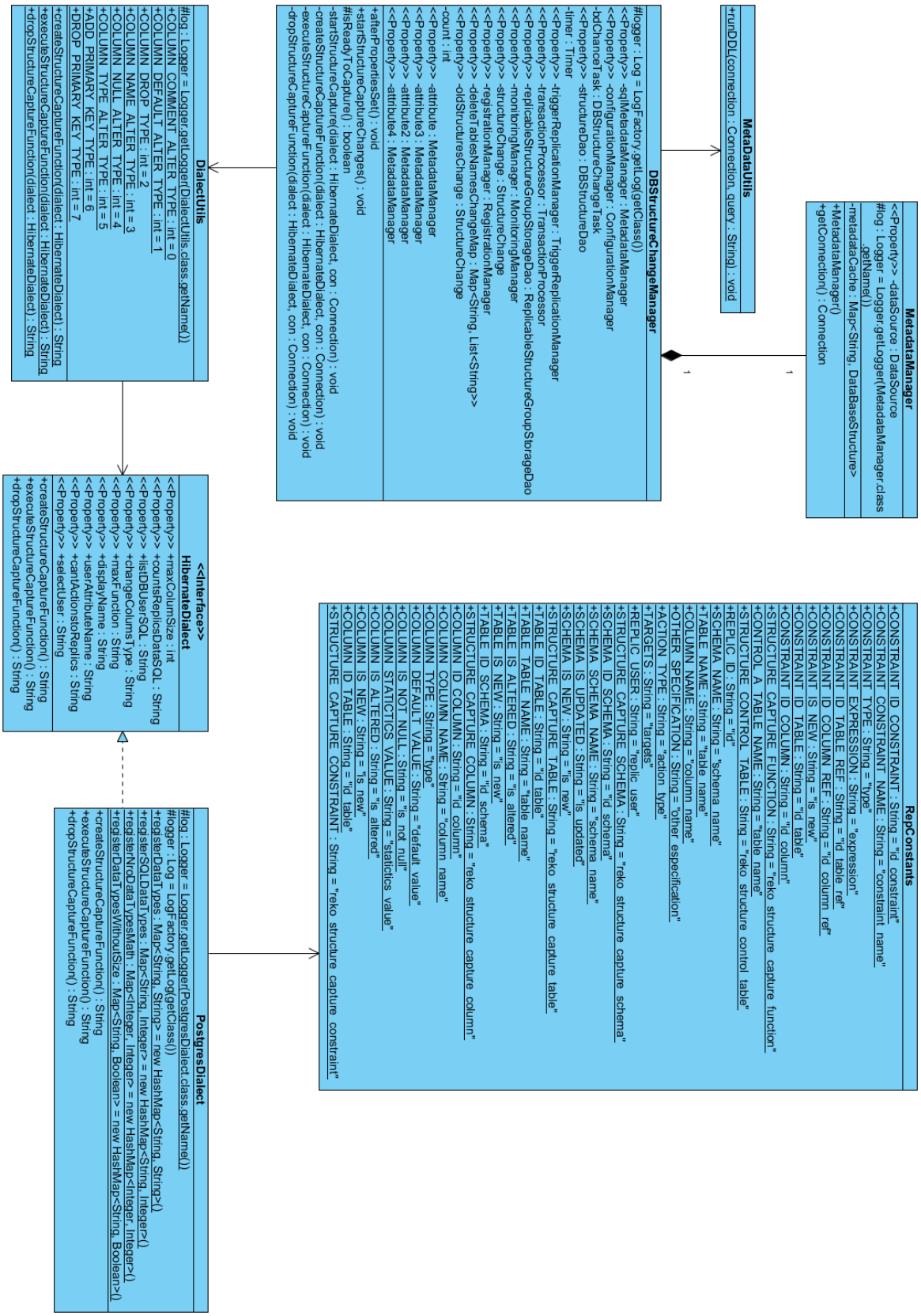

Figura 6: DC Capturar estructura inicial
Fuente: elaboración propia

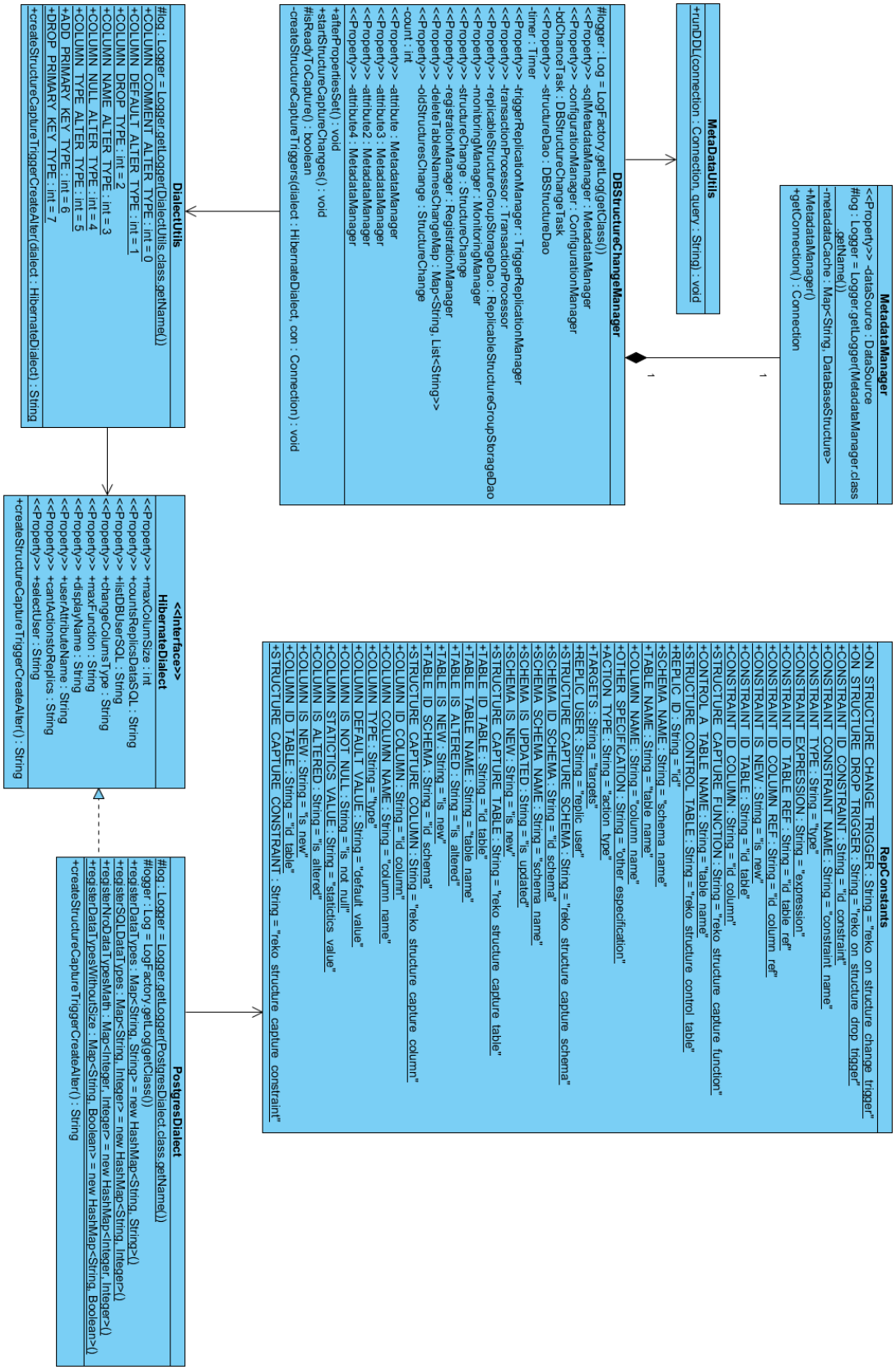

Figura 7: DC Crear *trigger* de captura de cambios de tipo CREATE _ ALTER
Fuente: elaboración propia

2.4.3 Descripción de las clases

A continuación, se realiza la descripción de algunas clases de la solución, para ver las restantes ver Anexos (3, 4):

Tabla 10: Descripción de la clase DBStructureChangeManager

Descripción de la clase DBStructureChangeManager	
Nombre: DBStructureChangeManager	
Tipo de clase: Controladora	
Atributo	**Tipo**
sqlMetadataManager	private
configurationManager	private
Responsabilidades	
Nombre: afterPropertiesSet() : void	
Descripción: Reajusta el proceso de captura de cambios de estructura cuando cambia la configuración de réplica	
Nombre: startStructureCaptureChanges() : void	
Descripción: Inicia el proceso de captura de cambios de estructura	
Nombre: createTablesOfGeneralControl(HibernateDialect dialect, Connection con) : void	
Descripción: Crea todas tablas de control general de cambios de estructura en la BD	
Nombre: createStructureControlTable(HibernateDialect dialect, Connection con): void	
Descripción: Crea la tabla de control de cambios de estructura en la BD	
Nombre: createTableCaptureSchema(HibernateDialect dialect, Connection con) : void	
Descripción: Crea la tabla de captura de estructura para los esquemas en la BD	
Nombre: createTableCaptureTable(HibernateDialect dialect, Connection con) : void	
Descripción: Crea la tabla de captura de estructura para las tablas en la BD	
Nombre: createTableCaptureColumn(HibernateDialect dialect, Connection con): void	
Descripción: Crea la tabla de captura de estructura para las columnas en la BD	
Nombre: createTableCaptureConstraint(HibernateDialect dialect, Connection con) : void	
Descripción: Crea la tabla de captura de estructura para las restricciones en la BD	
Nombre: stopStructureCaptureChanges(HibernateDialect dialect, Connection con) : void	
Descripción: Detiene el proceso de captura de cambios de estructura	
Nombre: dropStructureCaptureTriggers(HibernateDialect dialect, Connection con) : void	
Descripción: Elimina los *trigger*s de captura de cambios de estructura de la BD	
Nombre: startStructureCapture(HibernateDialect dialect, Connection con) : void	
Descripción: Realiza la captura inicial de la estructura de la BD	
Nombre: createStructureCaptureFunction(HibernateDialect dialect, Connection con) : void	
Descripción: Crea la función de captura inicial de estructura en la BD	
Nombre: executeStructureCaptureFunction(HibernateDialect dialect, Connection con): void	
Descripción: Ejecuta la función de captura inicial de estructura en la BD	
Nombre: dropStructureCaptureFunction(HibernateDialect dialect, Connection con): void	
Descripción: Elimina la función de captura inicial de estructura de la BD	
Nombre: createStructureCaptureTriggers(HibernateDialect dialect, Connection con): void	
Descripción: Crea los *triggers* de captura de cambios de estructura en la BD	

Fuente: elaboración propia

Tabla 11: Descripción de la clase DialectUtils

Descripción de la clase DialectUtils	
Nombre: DialectUtils	
Tipo de clase: Modelo	
Atributo	**Tipo**
-	-
Responsabilidades	
Nombre: dropStructureCaptureTriggers(HibernateDialect dialect): String	
Descripción: Devuelve la cadena SQL para eliminar los *triggers* de captura de cambios de estructura de la BD, según el tipo de dialecto	
Nombre: createStructureCaptureFunction(HibernateDialect dialect): String	
Descripción: Devuelve la cadena SQL para crear la función de captura de estructura inicial en la BD, según el tipo de dialecto	
Nombre: executeStructureCaptureFunction(HibernateDialect dialect): String	
Descripción: Devuelve la cadena SQL para ejecutar la función de captura de estructura inicial en la BD, según el tipo de dialecto	
Nombre: dropStructureCaptureFunction(HibernateDialect dialect): String	
Descripción: Devuelve la cadena SQL para eliminar la función de captura de estructura inicial de la BD, según el tipo de dialecto	
Nombre: createStructureCaptureTriggerCreateAlter(HibernateDialect dialect): String	
Descripción: Devuelve la cadena SQL para crear el *trigger* de captura de cambios de estructura de tipo CREATE / ALTER en la BD, según el tipo de dialecto	
Nombre: createStructureCaptureTriggerDrop(HibernateDialect dialect): String	
Descripción: Devuelve la cadena SQL para crear el *trigger* de captura de cambios de estructura de tipo DROP en la BD, según el tipo de dialecto	
Nombre: createStructureControlTable(HibernateDialect dialect): String	
Descripción: Devuelve la cadena SQL para crear la tabla de control de cambios de estructura en la BD, según el tipo de dialecto	
Nombre: createTableCaptureSchemas(HibernateDialect dialect): String	
Descripción: Devuelve la cadena SQL para crear la tabla de captura de estructura para los esquemas en la BD, según el tipo de dialecto	
Nombre: createTableCaptureTables(HibernateDialect dialect): String	
Descripción: Devuelve la cadena SQL para crear la tabla de captura de estructura para las tablas en la BD, según el tipo de dialecto	
Nombre: createTableCaptureColumns(HibernateDialect dialect): String	
Descripción: Devuelve la cadena SQL para crear la tabla de captura de estructura para las columnas en la BD, según el tipo de dialecto	
Nombre: createTableCaptureConstraints(HibernateDialect dialect): String	
Descripción: Devuelve la cadena SQL para crear la tabla de captura de estructura para las restricciones en la BD, según el tipo de dialecto	

Fuente: elaboración propia

Tabla 12: Descripción de la clase HibernateDialect

Descripción de la clase HibernateDialect	
Nombre: HibernateDialect	
Tipo de clase: Modelo <<*Interface*>>	
Atributo	**Tipo**
-	-
Responsabilidades	
Nombre: *dropStructureCaptureTriggers*(): String	
Descripción: Devuelve la cadena SQL para eliminar los *triggers* de captura de cambios de estructura de la BD, según el tipo de dialecto	
Nombre: *createStructureCaptureFunction*(): String	
Descripción: Devuelve la cadena SQL para crear la función de captura de estructura inicial en la BD, según el tipo de dialecto	
Nombre: *executeStructureCaptureFunction*(): String	
Descripción: Devuelve la cadena SQL para ejecutar la función de captura de estructura inicial en la BD, según el tipo de dialecto	
Nombre: *dropStructureCaptureFunction*(): String	
Descripción: Devuelve la cadena SQL para eliminar la función de captura de estructura inicial de la BD, según el tipo de dialecto	
Nombre: *createStructureCaptureTriggerCreateAlter*(): String	
Descripción: Devuelve la cadena SQL para crear el *trigger* de captura de cambios de estructura de tipo CREATE / ALTER en la BD, según el tipo de dialecto	
Nombre: *createStructureCaptureTriggerDrop*(): String	
Descripción: Devuelve la cadena SQL para crear el *trigger* de captura de cambios de estructura de tipo DROP en la BD, según el tipo de dialecto	
Nombre: *createStructureControlTableSQL*(): String	
Descripción: Devuelve la cadena SQL para crear la tabla de control de cambios de estructura en la BD, según el tipo de dialecto	
Nombre: *createCaptureSchemasTableSQL*(): String	
Descripción: Devuelve la cadena SQL para crear la tabla de captura de estructura para los esquemas en la BD, según el tipo de dialecto	
Nombre: *createCaptureTablesTableSQL*(): String	
Descripción: Devuelve la cadena SQL para crear la tabla de captura de estructura para las tablas en la BD, según el tipo de dialecto	
Nombre: *createCaptureColumnsTableSQL*(): String	
Descripción: Devuelve la cadena SQL para crear la tabla de captura de estructura para las columnas en la BD, según el tipo de dialecto	
Nombre: *createCaptureConstraintsTableSQL*(): String	
Descripción: Devuelve la cadena SQL para crear la tabla de captura de estructura para las restricciones en la BD, según el tipo de dialecto	

Fuente: elaboración propia

2.4.4 Diagrama de Despliegue (DD)

Un DD modela la arquitectura en tiempo de ejecución de un sistema. Esto muestra la configuración de los elementos de hardware (nodos) y muestra cómo los elementos y artefactos del software se trazan en esos nodos. [30]

En la figura 8 se muestra el diagrama de despliegue de Reko para el proceso de captura de cambios de estructura:

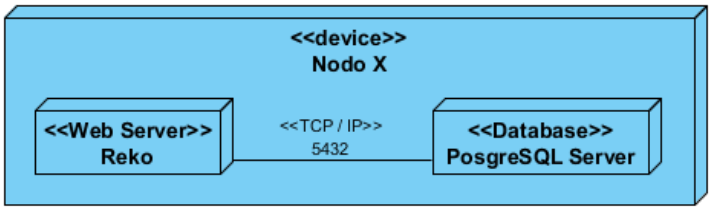

Figura 8: Diagrama de Despliegue
Fuente: elaboración propia

2.5 Conclusiones Parciales

El desarrollo de este capítulo permitió arribar a las siguientes conclusiones:

1. Se definieron y especificaron los cinco RF que darán cumplimiento al objetivo planteado una vez implementados, descritos en sus respectivas HU.
2. Se elaboró el modelo de diseño aplicando los patrones Experto, Controlador, Polimorfismo, Alta Cohesión, Bajo Acoplamiento y Fachada, lo que permitió obtener un modelo robusto.

Capítulo III: Implementación y Pruebas

A continuación, se muestran los diagramas de componentes que describen el proceso de implementación y el estilo de código empleado en el desarrollo de la solución. También se realizan pruebas al sistema para demostrar la validez del mismo.

3.1 Modelo de Implementación

El modelo de implementación describe cómo los elementos de diseño se implementan en componentes (ficheros de código fuente, de código binario, ejecutable, scripts), tomando el resultado del modelo de diseño para generar el código final del sistema. [31]

3.1.1 Diagrama de Componente

Un diagrama de componentes describe la descomposición física del sistema de software en componentes (código fuente, binario, ejecutable, librerías dinámicas, páginas web, interfaces, generalización, asociación y realización), además de la relación que existen entre ellos. [23]

A continuación, se muestra el diagrama de componentes correspondiente al subsistema Capturador de Cambios, con los principales componentes que intervienen en la implementación de la solución:

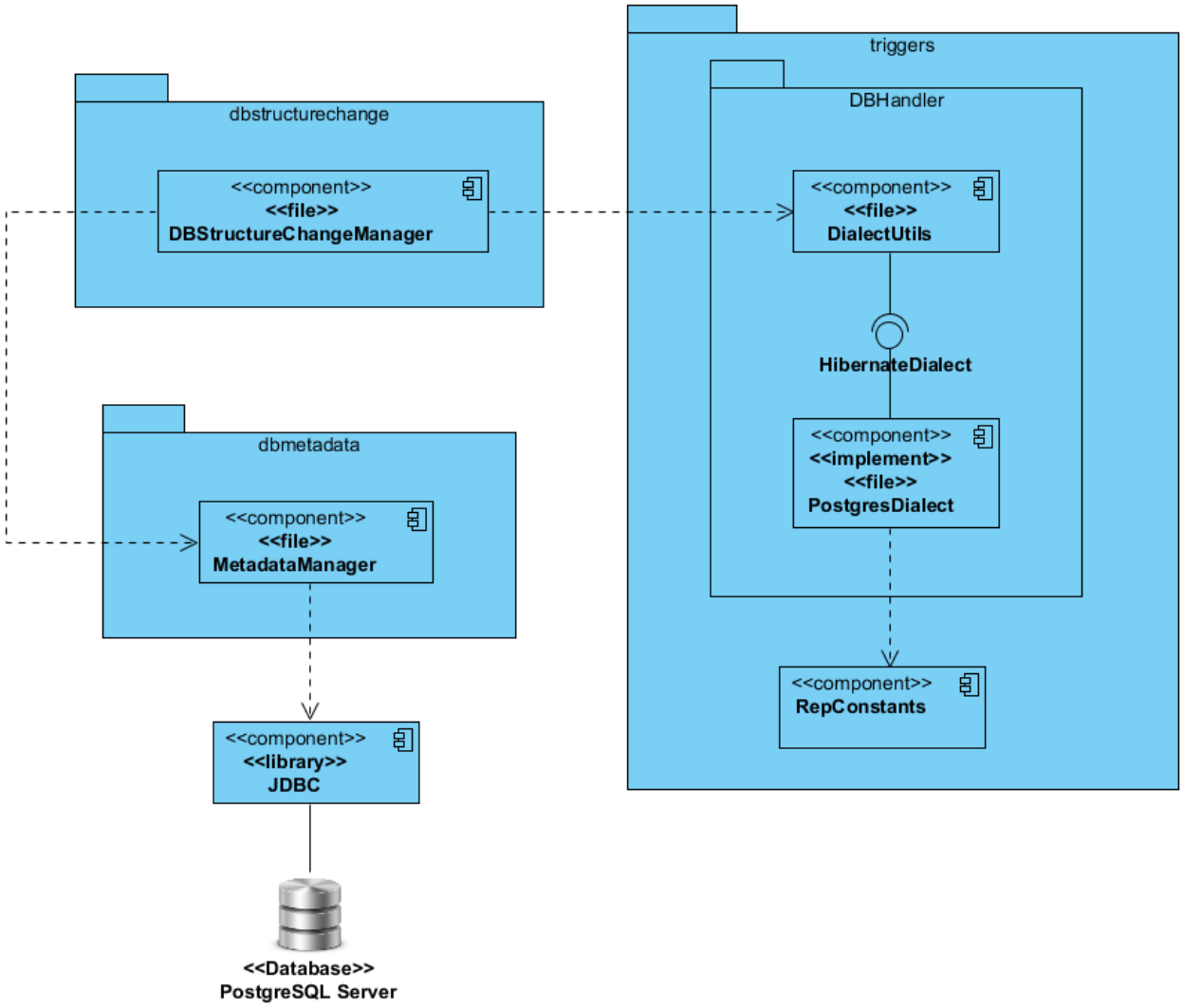

Figura 9: Diagrama de componente del subsistema capturador de cambios
Fuente: elaboración propia

3.1.2 Código Fuente

La implementación se basó principalmente en sentencias SQL para el trabajo con la BD y Java para la programación de Reko.

Estándar de codificación *java*

Las técnicas de codificación incorporan muchos aspectos del desarrollo del software. Aunque generalmente no afectan a la funcionalidad de la aplicación, sí contribuyen a una mejor compresión del código fuente. En esta fase se tienen en cuenta todos los tipos de código fuente, incluidos los lenguajes de programación, de secuencias de comandos, de marcado o de consulta. [32].

A continuación, se describe una parte del estándar de codificación empleado para implementar la solución, para consultar el resto ver Anexo (7):

Para los identificadores de variables y métodos se hace uso de la variante *lowerCamelCase*, empiezan con minúsculas y si están compuestos por varias palabras las siguientes empezarán con mayúscula.

Ejemplos:

```
Log logger;

MetadataManager sqlMetadataManager;
```

public void addCountries() {}

- **Métodos:**

Los nombres de método deben iniciar con un verbo. Ejemplo:

```
public void startStructureCaptureChanges(){…}
```

Los obtenedores de campos privados en las clases tienen el prefijo "get". Ejemplo:

```
public MetadataManager getSqlMetadataManager(){…}
```

Los modificadores de campos privados en las clases tienen el prefijo "set". Ejemplo:

```
public void setSqlMetadataManager(…){…}
```

Los obtenedores con el resultado de booleano tienen el prefijo "is". Ejemplo:

```
protected boolean isReadyToCapture(){…}
```

3.2 Evaluación del Diseño Aplicando Métricas de Software

Las métricas de software son medidas cuantitativas que permiten a los desarrolladores tener una visión profunda de la eficacia del proceso del software y de los proyectos que dirigen utilizando el proceso como un marco de trabajo. Se reúnen los datos básicos de calidad y productividad. Estos datos son entonces analizados, comparados con promedios anteriores, y evaluados para determinar las mejoras en la calidad y productividad. Las métricas son también utilizadas para señalar áreas con problemas de manera que se puedan desarrollar los remedios y mejorar el proceso del software. [23]

Se emplearon las métricas Tamaño Operacional de Clase y Relaciones entre Clases para evaluar los siguientes atributos de calidad en la implementación de la solución:

- **Responsabilidad**: consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta. Una buena distribución de la responsabilidad en las clases, permite que, en caso de fallo, ningún proceso sea demasiado crítico como para dejar fuera de servicio el sistema.
- **Complejidad de implementación**: consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- **Reutilización**: consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento**: consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.
- **Complejidad del mantenimiento**: consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirectamente, pero fuertemente en los costes y la planificación del proyecto.
- **Cantidad de pruebas**: consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (unidad) del producto (componente, módulo, clase, conjunto de clases, entre otras) diseñado. [23]

3.2.1 Tamaño Operacional de Clase (TOC)

Esta métrica se refiere a la cantidad de métodos que tiene una clase. [33] La tabla 13 muestra la relación que existe entre los atributos de calidad y la forma en que son afectados por el TOC.

Tabla 13: Tamaño Operacional de Clase

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

Fuente: [33]

Para los cuales están definidos los siguientes criterios y categorías de evaluación:

Tabla 14: Criterios de evaluación para la métrica TOC

Atributo	Categoría	Criterio
Responsabilidad	Baja	TOC <= Promedio[21]
	Media	Promedio < TOC <= 2*Promedio
	Alta	TOC > 2*Promedio
Complejidad de implementación	Baja	TOC <= Promedio
	Media	Promedio < TOC <= 2*Promedio
	Alta	TOC > 2*Promedio
Reutilización	Baja	TOC > 2*Promedio
	Media	Promedio < TOC <= 2*Promedio
	Alta	TOC <= Promedio

Fuente: [33]

Resultados obtenidos en la aplicación de la métrica TOC

Luego de aplicar la métrica TOC, se puede comprobar que el modelo de diseño propuesto tiene una calidad aceptable, debido a que el comportamiento de los atributos de calidad es positivo. Ver Anexo (5) para una descripción más detallada.

La figura 10 muestra que un 67% de las clases posee una **responsabilidad** baja y sólo un 22% una responsabilidad alta, por lo que, en caso de fallo, ningún proceso es demasiado crítico como para dejar fuera de servicio al sistema.

La figura 11 describe resultados similares, pero evaluando la **complejidad de implementación**, demostrando que el sistema no es difícil de implementar.

La figura 12 muestra que un 67% de las clases tiene un alto grado de **reutilización** y sólo un 22% una reutilización baja; por lo que las funcionalidades pueden ser aprovechadas al máximo.

[21] Promedio: Se refiere al promedio de la cantidad de métodos por clase.

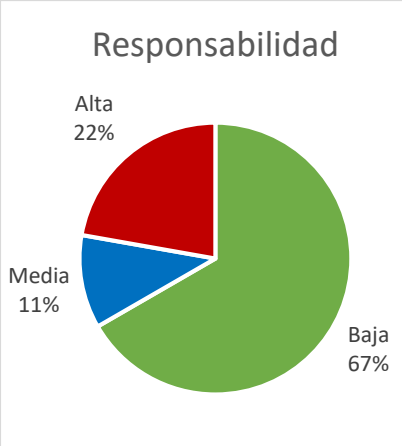

Figura 10: Evaluación de la Responsabilidad mediante TOC
Fuente: elaboración propia

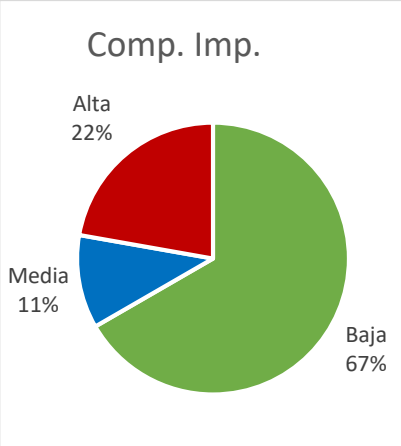

Figura 11: Evaluación de la Complejidad de Implementación mediante TOC
Fuente: elaboración propia

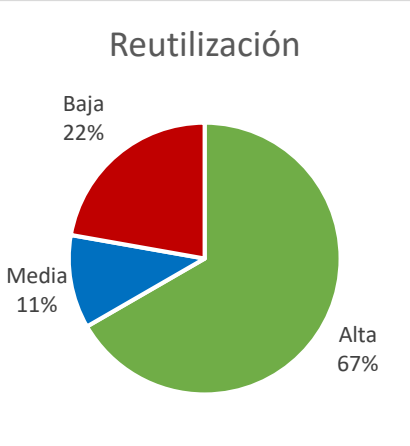

Figura 12: Evaluación de la Reutilización mediante TOC
Fuente: elaboración propia

3.2.2 Relaciones entre Clases (RC)

Esta métrica se refiere a la cantidad de relaciones de uso que tiene una clase con otras. [33] La tabla 15 muestra la relación que existe entre los atributos de calidad y la forma en que son afectados por las RC.

Tabla 15: Relaciones entre clases

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

Fuente: [33]

Para los cuales están definidos los siguientes criterios y categorías de evaluación:

Tabla 16: Criterios de evaluación para la métrica RC

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	RC = 0
	Baja	RC = 1
	Media	RC = 2
	Alta	RC > 2
Complejidad de mantenimiento	Baja	RC <= Promedio[22]
	Media	Promedio < RC <= 2*Promedio
	Alta	RC > 2*Promedio
Reutilización	Baja	RC > 2*Promedio
	Media	Promedio < RC <= 2*Promedio
	Alta	RC <= Promedio
Cantidad de pruebas	Baja	RC <= Promedio
	Media	Promedio < RC <= 2*Promedio
	Alta	RC > 2*Promedio

Fuente: [33]

Resultados obtenidos en la aplicación de la métrica RC

Luego de aplicar la métrica RC, se puede comprobar que el modelo de diseño propuesto tiene una calidad aceptable, debido a que el comportamiento de los atributos de calidad también es positivo, al igual que cuando se aplica la métrica TOC. Ver Anexo (6) para una descripción más detallada.

La figura 13 muestra que un 45% de las clases posee un nivel de **acoplamiento** bajo y un 33% no tiene, por lo que sólo un 22% de las clases posee un nivel medio o alto; demostrando así, que el grado de dependencia entre las clases del modelo de diseño planteado es bajo y el impacto de cualquier cambio en una clase es bajo sobre el resto del sistema.

La figura 14 especifica que el 78% de las clases tiene una baja **complejidad de mantenimiento**, por lo que no se requiere de gran esfuerzo para realizarle mejoras o correcciones al sistema en general.

La figura 15 detalla que un 78% de las clases posee una alta **reutilización**, dejando sólo un 22% de clases cuya reutilización es media o baja; por lo que se demuestra que el modelo de diseño propuesto, permite un elevado aprovechamiento de sus funcionalidades, al igual que cuando se aplicó la métrica TOC.

[22] Promedio: Se refiere al promedio de la cantidad de relaciones de uso de una clase con otras.

La figura 16 muestra que sólo a un 11% de las clases tiene el valor de **cantidad de pruebas** alto, por lo que no se necesita gran esfuerzo para realizar pruebas al sistema en general.

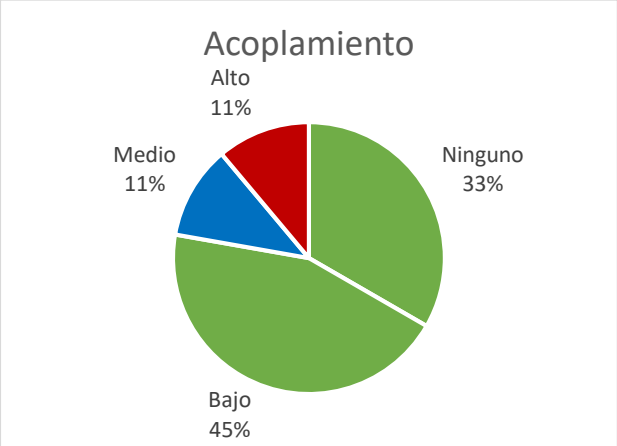

Figura 13: Evaluación del Acoplamiento mediante RC
Fuente: elaboración propia

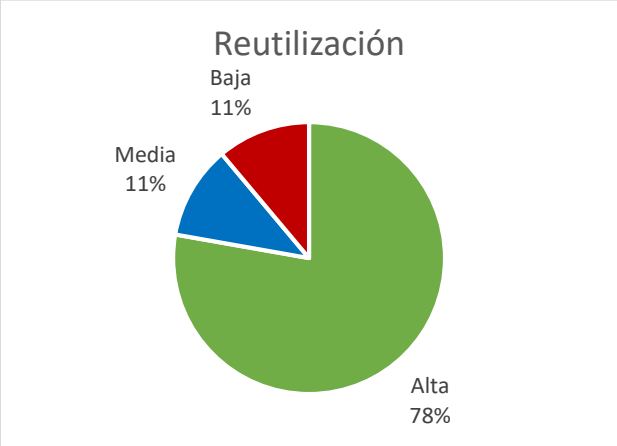

Figura 15: Evaluación de la Reutilización mediante RC
Fuente: elaboración propia

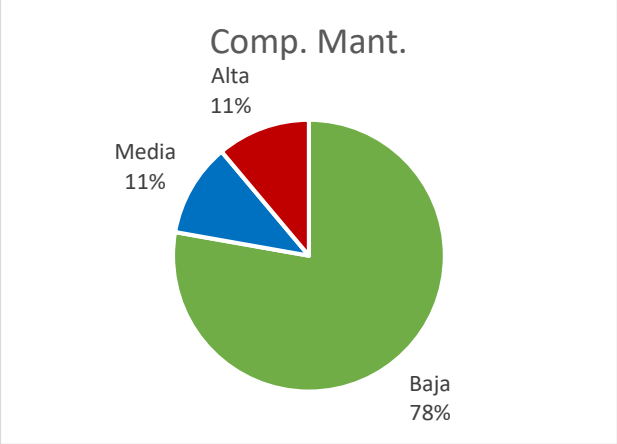

Figura 14: Evaluación de la Complejidad de mantenimiento mediante RC
Fuente: elaboración propia

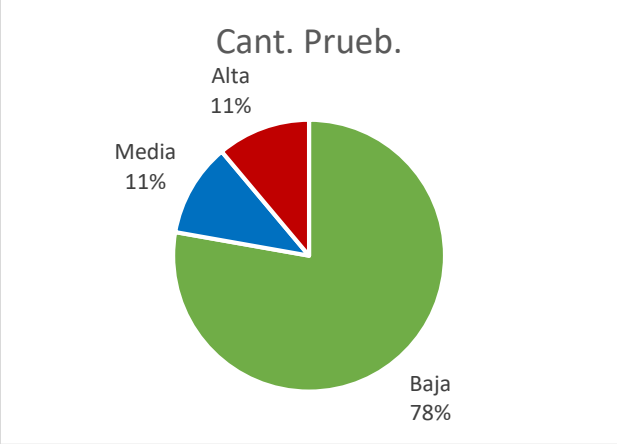

Figura 16: Evaluación de la Cantidad de Pruebas mediante RC
Fuente: elaboración propia

3.2.3 Matriz de inferencia de indicadores de calidad

La matriz inferencia de indicadores de calidad, también llamada matriz de cubrimiento, es una representación estructurada de los atributos de calidad y métricas utilizadas para evaluar la calidad del diseño propuesto. Dicha matriz permite conocer si los resultados obtenidos de las relaciones atributo/métrica es positivo o no, llevando estos resultados a una escalabilidad numérica donde, si los resultados son positivos se le asigna el valor de 1, si son negativos toma

valor 0 y si no existe relación es considerada como nula y es representada con un guión simple (-). Luego se puede obtener un resultado general para cada atributo promediando todas sus relaciones no nulas. [33]

La tabla 17 muestra que los resultados obtenidos para cada atributo de calidad medido son positivos:

Tabla 17: Matriz de Interferencia de Indicadores de Calidad

Atributos\Métricas	TOC	RC	Promedio
Responsabilidad	1	-	1
Complejidad de implementación	1	-	1
Reutilización	1	1	1
Acoplamiento	-	1	1
Complejidad de mantenimiento	-	1	1
Cantidad de Pruebas	-	1	1

Fuente: elaboración propia

3.2.4 Resultados obtenidos en la aplicación de las métricas de software

Luego de aplicar las métricas TOC y RC, se puede comprobar que el modelo de diseño propuesto tiene una calidad aceptable, debido a que el comportamiento de los atributos de calidad es positivo:

- El grado de **responsabilidad** del sistema es bajo, por lo que ningún proceso es demasiado crítico como para dejar fuera de servicio al sistema en caso de fallo.
- La **complejidad de implementación** del sistema es baja.
- El sistema posee un alto grado de **reutilización**, por lo que sus funcionalidades pueden ser aprovechadas al máximo.
- Las clases del sistema poseen un bajo nivel de **acoplamiento**, reduciendo las dependencias entre clases y permitiendo realizar cambios sin afectar en gran medida al resto de sistema.
- La **complejidad de mantenimiento** es baja, por lo que no se requiere de gran esfuerzo para realizarle mejoras o correcciones al sistema en general.
- No se necesita gran esfuerzo para realizar **pruebas** al sistema en general.

3.3 Pruebas de Software

Las pruebas son una actividad, en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos específicos, los resultados son observados y registrados para una posterior evaluación. Son un elemento de suma importancia para la calidad del software, por lo que se llevan a cabo durante todo el ciclo de vida del mismo. Constituyen una revisión final de las especificaciones del diseño y de la codificación. [34]

Para el desarrollo de las pruebas se tienen en cuenta un conjunto de estrategias a seguir, y de esta forma lograr la calidad requerida y el cumplimiento de los objetivos. La estrategia de prueba que elige la mayor parte de los equipos de software se ubica entre estos dos extremos. Toma un enfoque incremental de las pruebas; inicia con las pruebas de unidades individuales del programa, pasa a pruebas diseñadas para facilitar la integración de las unidades, y culmina con pruebas que se realizan sobre el sistema construido. [34]

Para alcanzar el objetivo de las pruebas de software es necesario planear y ejecutar una serie de pasos (pruebas de unidad, integración, validación y sistema). Las pruebas de unidad e integración se concentran en la verificación funcional de cada componente y en la incorporación de componentes en la arquitectura del software. La prueba de validación demuestra el cumplimiento de los requisitos del software y la prueba del sistema valida el software una vez que sea incorporado a un sistema mayor. [34]

Con el fin de validar la solución implementada, se prepararon y llevaron a cabo un conjunto de pruebas, las cuales se pueden clasificar en pruebas de unicidad y de aceptación.

3.3.1 Pruebas de Unicidad

Las pruebas de unicidad están enfocadas a los elementos más pequeños del *software*. Aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera. La prueba de unidad está orientada a la técnica de prueba caja blanca. [34]

Pruebas de Caja Blanca

La prueba de caja blanca, en ocasiones llamada prueba de caja de cristal, es un método de diseño que usa la estructura de control descrita como parte del diseño al nivel de componentes para derivar los casos de prueba. Al emplear los métodos de prueba de caja blanca, el ingeniero del software podrá derivar casos de prueba que:

- Garanticen que todas las rutas independientes dentro del módulo se han ejercitado por lo menos una vez.
- Ejerciten los lados verdaderos y falsos de todas las decisiones lógicas.
- Ejecuten todos los bucles en sus límites y dentro de sus límites operacionales.
- Ejerciten estructuras de datos internos para asegurar su validez. [34]

Para la realización de las pruebas de caja blanca, se opta por la técnica de camino básico, pues permite determinar por adelantado, el número de pruebas mínimas necesarias.

Técnica de camino básico

Esta técnica permite obtener una medida de la complejidad lógica de un diseño procedimental y utiliza esa medida como guía para la definición de un conjunto básico de caminos de ejecución, de los cuales se obtienen los casos de prueba, que garantizan la ejecución de cada sentencia del programa al menos una vez, durante las pruebas. [34]

Su ejecución se describe en cuatro pasos:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado: se utiliza para representar el flujo de control lógico de un programa.
2. Se calcula la complejidad ciclomática del grafo: el valor calculado define el número de rutas independientes en el conjunto básico de un programa, y proporciona un límite superior para el número de pruebas que deben aplicarse, lo cual asegura que todas las instrucciones se hayan ejecutado por lo menos una vez.
3. Se determina un conjunto básico de caminos independientes: es cualquier ruta del programa que ingresa por lo menos un nuevo conjunto de instrucciones de procesamiento o una nueva condición.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico. [34]

A continuación, se muestra la aplicación de la técnica de camino básico, al método `createStructureCaptureTriggerDrop()` de la clase `PostgresDialect`, el cual tiene como objetivo, devolver la consulta para PostgreSQL que creará el trigger DDL para la captura de cambios de tipo DROP en la BD. Se escoge este método, debido a su elevada importancia para el proceso de captura de cambios de estructura.

Tabla 18: Definición de los nodos de flujo en el método `createStructureCaptureTriggerDrop()`

Nodo	Código

0	`public String createStructureCaptureTriggerDrop(…) {`
	`StringBuffer query = new StringBuffer();`
	`query.append("DROP EVENT TRIGGER IF EXISTS " +`
	`RepConstants.ON_STRUCTURE_DROP_TRIGGER + ";\n\n");`
	`query.append("CREATE OR REPLACE FUNCTION " +`
	`RepConstants.ON_STRUCTURE_DROP_TRIGGERED_FUNCTION +`
	`"()\n" + "RETURNS event_trigger\n" + "LANGUAGE plpgsql\n"`
	`+ "AS $$\n" + "DECLARE\n");`
	`query.append(" reko_replic_user TEXT := (SELECT current_user`
	`FROM user);\n\n");`
	`query.append(" obj RECORD;\n");`
	`query.append("BEGIN\n");`
	`query.append(" --INICIO CHEQUEO DE OBJECT_TYPE\n"`
	`+ " FOR obj IN SELECT * FROM`
	`pg_event_trigger_dropped_objects()\n" + "LOOP\n\n");`
	`query.append(" IF obj.object_type = 'schema' AND`
	`obj.object_name NOT LIKE 'reko%'\n" + " THEN\n");`

1, 2	`for (int i = 0; i < targets.size(); i++) {`

3	`String target = targets.get(i);`
	`String[] users = allowedChangesUsers.get(i).get(2);`
	`boolean existsTargets = !target.isEmpty();`
	`boolean isDropSchemaAllowed = allowedChanges.get(i)[2];`
	`boolean existsAllowedUserForDrops = users != null;`

4	`if (existsTargets && isDropSchemaAllowed &&` `existsAllowedUserForDrops) {`
5	`String user = "'" + users[0] + "'";`
6, 7	`for (int u = 1; u < users.length; u++) {`
8	`user += ", '" + users[u] + "'";`
	`}`

9	`query.append(" IF reko_replic_user IN(" +`
	`user + ") THEN\n\n");`
	`query.append(" INSERT INTO " +`
	`RepConstants.STRUCTURE_CONTROL_TABLE + " ("`
	`+ RepConstants.ACTION_TYPE + ", " +`
	`RepConstants.SCHEMA_NAME + ", " +`
	`RepConstants.REPLIC_USER`
	`+ ", " + RepConstants.TARGETS + ")\n" + "`
	`VALUES ('"+ ActionType.drop_schema.toString() + "',`
	`obj.object_name, reko_replic_user, '" + target`
	`+ "');\n\n");`
	`query.append(" END IF;\n");`

	`}` `}`

10	`query.append(" DELETE FROM " +`
	`RepConstants.STRUCTURE_CAPTURE_SCHEMA + " cs WHERE`
	`obj.objid = cs." + RepConstants.SCHEMA_ID_SCHEMA +`
	`";\n");`
	`query.append(" ELSIF NOT obj.original AND`
	`obj.schema_name NOT IN (SELECT ns.nspname FROM`
	`pg_catalog.pg_namespace ns WHERE ns.nspname =`
	`obj.schema_name)\n" + " THEN\n" + "--No hacer`
	`nada\n");`

```
        query.append("          ELSIF obj.object_type = 'table' AND
              obj.object_name NOT LIKE 'reko%' AND obj.schema_name NOT
              LIKE 'reko%'\n" + "           THEN\n");
11, 12  for (int i = 0; i < targets.size(); i++) {
13              String target = targets.get(i);
                boolean existsTargets = !target.isEmpty();
14              if (existsTargets) {
15                      List<String> schemas = schemaNames.get(i);
                        List<boolean[]> internalChanges =
                              allowedInternalChanges.get(i);
                        List<List<String[]>> internalUsers =
                              allowedInternalChangesUsers.get(i);
16, 17                  for (int j = 0; j < schemas.size(); j++) {
18                              String[] users = internalUsers.get(j).get(2);
                                boolean isDropAllowed =
                                      internalChanges.get(j)[2];
                                boolean existsAllowedUserForDrops = users !=
                                      null;
19                              if (isDropAllowed &&
                                      existsAllowedUserForDrops) {
20                                      String user = "'" + users[0] + "'";
21, 22                                  for (int u = 1; u < users.length; u++){
23                                              user += ", '" + users[u] + "'";
                                        }
24                                      query.append(" IF obj.schema_name = '"
                                              + schemas.get(j) + "' AND
                                              reko_replic_user IN (" + user +
                                              ") THEN\n\n");
                                        query.append("    INSERT INTO " +
                                              RepConstants.
                                              STRUCTURE_CONTROL_TABLE + " (" +
                                              RepConstants.ACTION_TYPE + ", " +
                                              RepConstants.SCHEMA_NAME + ", "
                                              + RepConstants.TABLE_NAME + ", "
                                              + RepConstants.REPLIC_USER + ", "
                                              + RepConstants.TARGETS + ")\n" +
                                              "VALUES ('" +ActionType.
                                              drop_table.toString()+ "',
                                              obj.schema_name, obj.object_name,
                                              reko_replic_user, '" + target +
                                              "');\n\n");
                                        query.append("END IF;\n");
                                }
                        }
                }
        }
25      query.append("                DELETE FROM " +
              RepConstants.STRUCTURE_CAPTURE_TABLE + " ct WHERE
              obj.objid = ct."+ RepConstants.TABLE_ID_TABLE + ";\n\n");
        query.append("          END IF;\n\n");
        query.append("    END LOOP;\n" + "  --FIN CHEQUEO DE OBJECT_TYPE
              \n\n");
        query.append("END;\n" + "$$;\n\n" + "CREATE EVENT TRIGGER " +
              RepConstants.ON_STRUCTURE_DROP_TRIGGER + " ON sql_drop\n"
              + "EXECUTE PROCEDURE " + RepConstants.
```

```
                ON_STRUCTURE_DROP_TRIGGERED_FUNCTION + "();");
                return query.toString();
}
```

Fuente: elaboración propia

La figura 17 muestra cómo queda conformado gráficamente el flujo de control lógico del método anterior.

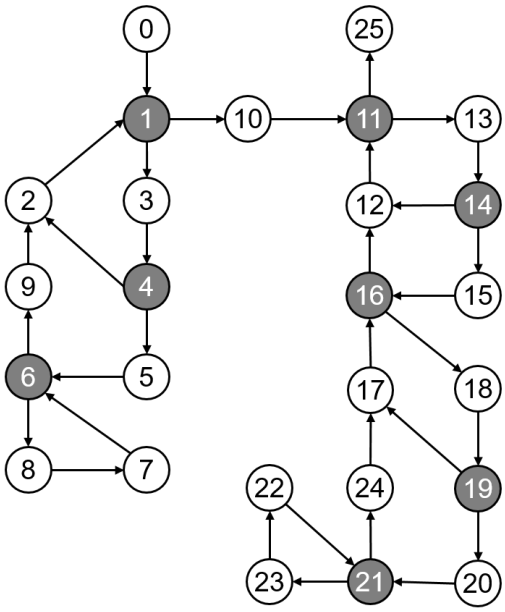

Figura 17: Flujo de control lógico correspondiente al método `createStructureCaptureTriggerDrop()`

Fuente: elaboración propia

Una vez construido el grafo, se procede a calcular la complejidad ciclomática usando las tres fórmulas descritas a continuación; las cuales devuelven el mismo resultado, asegurando que el cálculo es correcto.

Tabla 19: Complejidad ciclomática correspondiente al método `createStructureCaptureTriggerDrop()`

V(G) = R	V(G) = A - N + 2	V(G) = P + 1
R = 9 (regiones del grafo)	A = 33 (aristas) N = 26 (nodos)	P = 8 (nodos predicados)
	V(G) = 33 - 26 + 2	V(G) = 8 + 1
V(G) = 9	**V(G) = 9**	**V(G) = 9**

Fuente: elaboración propia

Conociendo que **V(G) = 9**, se puede afirmar que se necesitan hasta 9 caminos distintos para recorrer todos los nodos de grafo, donde cada camino representa una prueba a realizar para que

cada sentencia de código se ejecute al menos una vez. A continuación, se muestran los caminos determinados en la aplicación

1) 0, 1, 10, 11, 25
2) 0, 1, 3, 4, 2, 1, 10, 11, 13, 14, 12, 11, 25
3) 0, 1, 3, 4, 2, 1, 10, 11, 13, 14, 15, 16, 12, 11, 25
4) 0, 1, 3, 4, 5, 6, 9, 2, 1, 10, 11, 13, 14, 12, 11, 25
5) 0, 1, 3, 4, 5, 6, 8, 7, 6, 9, 2, 1, 10, 11, 13, 14, 12, 11, 25
6) 0, 1, 3, 4, 2, 1, 10, 11, 13, 14, 15, 16, 18, 19, 17, 16, 12, 11, 25
7) 0, 1, 3, 4, 2, 1, 10, 11, 13, 14, 15, 16, 18, 19, 20, 21, 24, 17, 16, 12, 11, 25
8) 0, 1, 3, 4, 2, 1, 10, 11, 13, 14, 15, 16, 18, 19, 20, 21, 23, 22, 21, 24, 17, 16, 12, 11, 25
9) 0, 1, 3, 4, 5, 6, 8, 7, 6, 9, 2, 1, 10, 11, 13, 14, 15, 16, 18, 19, 20, 21, 24, 17, 16, 12, 11, 25

A cada uno de estos caminos le corresponde un caso de prueba, como se muestra en la tabla 20:

Tabla 20: Casos de prueba por camino básico

Camino	Entrada	Resultado
1	No hay configuraciones de réplica establecidas	Se crea el trigger que captura cambios de tipo DROP, pero no registra los cambios en la tabla de control
2	Existe al menos una configuración de réplica establecida, pero no se le asignó a ningún nodo como destino	Se crea el trigger que captura cambios de tipo DROP, pero no registra los cambios en la tabla de control
3	Existe al menos una configuración de réplica con el destino correcto, pero no se habilitó la opción de eliminar esquema ni la opción de eliminar elementos a ningún esquema	Se crea el trigger que captura cambios de tipo DROP, pero no registra los cambios en la tabla de control
4	Existe al menos una configuración de réplica con el destino correcto, se habilitó la opción de eliminar esquema para un usuario, pero no la opción de eliminar elementos a ningún esquema	Se crea el trigger que captura cambios de tipo DROP, pero sólo registra las eliminaciones de esquemas realizadas por el usuario especificado en la tabla de control
5	Existe al menos una configuración de réplica con el destino correcto, se habilitó la opción de eliminar esquema para varios usuarios, pero no la opción de eliminar elementos a ningún esquema	Se crea el trigger que captura cambios de tipo DROP, pero sólo registra las eliminaciones de esquemas realizadas por los usuarios especificados en la tabla de control
6	Existe al menos una configuración de réplica con el destino correcto, se	Se crea el trigger que captura cambios de tipo DROP, pero no registra los cambios en la tabla de control

	habilitaron las opciones de eliminar esquema y eliminar elementos de esquemas, pero no se asignaron usuarios	
7	Existe al menos una configuración de réplica con el destino correcto, se habilitó la opción de eliminar elementos de un esquema para un usuario, pero no se habilitó la opción de eliminar esquemas	Se crea el trigger que captura cambios de tipo DROP, pero sólo registra las eliminaciones de elementos en el esquema realizadas por el usuario especificado
8	Existe al menos una configuración de réplica con el destino correcto, se habilitó la opción de eliminar elementos de un esquema para varios usuarios, pero no se habilitó la opción de eliminar esquemas	Se crea el trigger que captura cambios de tipo DROP, pero sólo registra las eliminaciones de elementos en el esquema realizadas por los usuarios especificados
9	Existe al menos una configuración de réplica con el destino correcto, se habilitó la opción de eliminar elementos de un esquema para un usuario, y se habilitó la opción de eliminar esquemas para varios usuarios	Se crea el trigger que captura cambios de tipo DROP, pero sólo registra las eliminaciones de elementos en el esquema realizadas por el usuario y las eliminaciones de esquemas por los usuarios especificados

Fuente: elaboración propia

Resultados de las pruebas de caja blanca

Las pruebas de caja blanca aplicadas al código fuente de la solución, utilizando el método de camino básico, demuestran que los Requisitos Funcionales están correctamente implementados y que el sistema responde como se espera ante determinadas acciones.

Fueron necesarias tres iteraciones de pruebas hasta alcanzar que el sistema estuviera listo, en la primera iteración se detectaron las siguientes no conformidades (NC):

- No se capturan todos los cambios configurados.
- No se especifica el usuario que realiza los cambios en la captura.
- Se registran cambios de estructura capturados en la tabla de control que no fueron configurados
- Se registran los cambios de estructura capturados en la tabla de control, pero no se especifica el destino de esos cambios.

En la segunda iteración fueron resueltas tres NC y se detectó otra:

- No se capturan todos los cambios configurados.

- Se registran los cambios de estructura capturados en la tabla de control para usuarios no permitidos en la configuración.

En la tercera iteración de pruebas fueron resueltas la NC y no se detectaron nuevas. La figura 18 muestra la relación entre las NC detectadas y resueltas por cada iteración de pruebas:

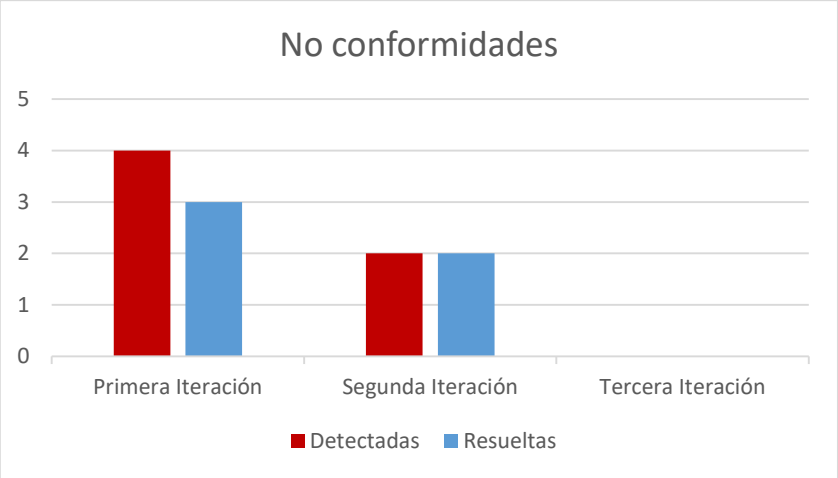

Figura 18: Relación de no conformidades en las pruebas de caja blanca, Fuente: elaboración propia

3.3.2 Pruebas de Aceptación

Las pruebas de aceptación del usuario es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido. [34]

Caja negra

Las pruebas de caja negra, también denominadas, pruebas de comportamiento se concentran en los requisitos funcionales del software, llevándose a cabo en la interfaz del software. El objetivo de esta prueba es demostrar que las funciones del software son operativas, que las entradas se aceptan y producen los resultados esperados. Las pruebas de caja negra tratan de encontrar errores en las siguientes categorías: [34]

- Funciones incorrectas o faltantes.
- Errores de interfaz.
- Errores de estructura de datos o en acceso a base de datos externas.
- Errores de comportamiento o desempeño.

- Errores de inicialización y término. [34]

Para realizar las pruebas de caja negra se eligió el método de partición equivalente, el cual permite realizar un conjunto de pruebas y obtener el máximo de los errores presentes en el proceso.

Método de partición equivalente

La partición equivalente es un método de prueba de caja negra que divide el campo de entrada en clases de datos que tienden a ejecutar determinadas funciones del software de las que pueden derivarse casos de prueba. Permite examinar los valores válidos y no válidos de las entradas existentes en el software. La partición equivalente se esfuerza por definir un caso de prueba que descubra ciertas clases de errores y reduce el número total de casos de prueba que deben desarrollarse. [34]

La tabla 21 describe la aplicación de este método al RF2, mostrando resultados satisfactorios:

Tabla 21: Resultados de la prueba uno de caja negra sobre la HU 2

Caso de Prueba	
Código: SC_1	**Historia de usuario**: 2
Nombre: Capturar estructura inicial	
Descripción: prueba para la funcionalidad de capturar estructura inicial.	
Condiciones de Ejecución: • El usuario debe estar autenticado. • Establecer la configuración de BD para PostgreSQL en el Módulo de Configuración General.	
Respuesta del Sistema/Flujo Central: • Al seleccionar la opción "Replicar cambios de esquemas" el sistema muestra un panel que brinda la posibilidad de iniciar el envío de los cambios de estructuras. • Al seleccionar el botón "Guardar" el sistema almacena la configuración con los parámetros establecidos, crea las tablas de control en la BD en caso de no existir, inicia la captura de los cambios de estructura si existe una configuración de réplica de estructura y muestra la configuración establecida.	
Resultado Esperado: el sistema realiza la captura de la estructura inicial almacenándola en las tablas de control.	

Evaluación de la Prueba: bien.

Fuente: elaboración propia

Resultados de las pruebas de caja negra

Fueron necesarias dos iteraciones para alcanzar un estado correcto de sistema implementado. En la primera iteración se detectaron las siguientes NC:

- Tipos de datos incompatibles entre los atributos OID de PostgreSQL los campos ID de las tablas de control de Reko.
- La captura de la estructura de la BD no se realiza cuando inicia Reko.
- La captura de la estructura de la BD no se realiza cuando cambia la configuración de réplica.

En la segunda iteración fueron resueltas y no se detectaron nuevas NC.

3.3.3 Análisis del Consumo de Recursos del Sistema

Una vez alcanzado un estado de implementación libre de errores, se procede a monitorear el consumo de recursos del sistema. Dicho análisis se realizó durante dos horas por cinco días consecutivos, permitiendo detectar que los parámetros chequeados se comportaron siempre de la misma forma, como se explica a continuación:

La tabla 22 muestra una relación de consumo de memoria del servidor de PostgreSQL mientras trabaja con Reko en su versión 4.0 y mientras trabaja con la solución implementada con *triggers DDL*, ambos casos durante periodos en los que la BD no sufrió cambios de estructura.

Tabla 22: Comparación del consumo de memoria

	Reko 4.0	Solución propuesta
Inicio	0.8 MB	0.8 MB
5 min	1.33 GB	5.4 MB
10 min	2.35 GB	5.3 MB
20 min	5.32 GB	5.5 MB
30 min	> 10.0 GB	5.5 MB
45 min	-	5.4 MB
60 min	-	5.4 MB
90 min	-	5.3 MB
120 min	-	5.4 MB

Fuente: elaboración propia

La figura 19 muestra la relación de espacio en disco ocupado por los *logs* de PostgreSQL al usar Reko 4.0 y al usar la solución implementada; ejecutando ambas versiones durante dos horas cada día en diferentes ordenadores. En ambos casos se aplica el mismo grupo de cambios de estructura, donde se puede apreciar que en solo 5 días casi se alcanzan los 2000 MB empleando la versión 4.0 de Reko.

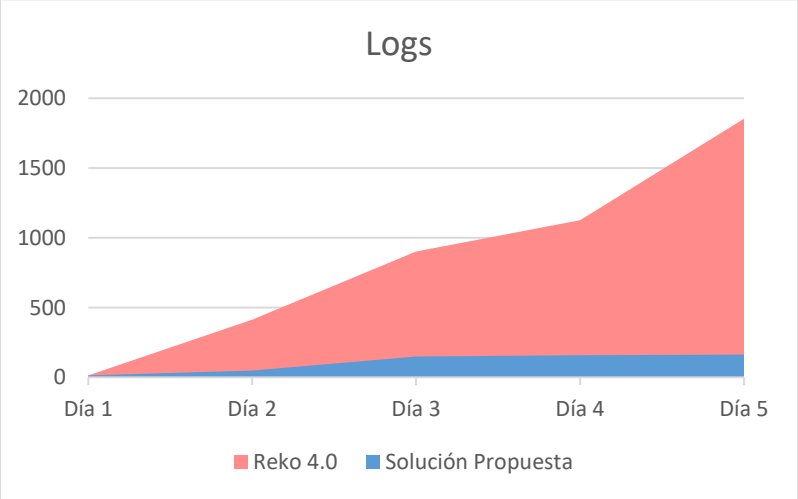

Figura 19: Espacio en disco ocupado por los *logs* de PostgreSQL
Fuente: elaboración propia

3.4 Conclusiones Parciales

1. En este capítulo se elaboró el modelo de implementación donde se describe el estándar de codificación usado para implementar la solución.
2. Se aplicaron las métricas de software TOC y RC que validaron la calidad del modelo de diseño propuesto como aceptable.
3. Se realizaron pruebas de software que permitieron detectar y resolver las no conformidades, así como validar el cumplimiento de los requisitos funcionales definidos.
4. Quedó demostrado mediante análisis, que el consumo de recursos del sistema se redujo considerablemente con respecto a la versión 4.0 de Reko.

Conclusiones

Con el desarrollo de la presente investigación se arriba a las siguientes conclusiones:

1. El empleo de la metodología, tecnologías, lenguajes y herramientas definidas para la investigación, permitió obtener un producto acorde los estándares de desarrollo del Replicador de Datos Reko.
2. Quedó implementado en Reko un nuevo mecanismo de captura de cambios de estructura para bases de datos gestionadas en PostgreSQL que emplea *triggers DDL* que contribuye a evitar conflictos en la réplica de datos y reduce el uso de recursos del sistema.
3. La realización de pruebas a la solución y la aplicación de métricas de software, permitieron detectar y corregir a tiempo y con el menor esfuerzo los errores existentes, obteniéndose un producto funcional y con la calidad requerida.

Recomendaciones

Partiendo de los resultados obtenidos se recomienda:

- Brindar a Reko la funcionalidad de captura de cambios de estructura para los gestores *MySQL* y *Microsoft SQL Server*.
- Extender la variedad de cambios de estructura que se pueden capturar para PostgreSQL.

Referencias Bibliográficas

1. SIERRA, Manuel. *¿QUÉ ES UNA BASE DE DATOS Y CUÁLES SON LOS PRINCIPALES TIPOS?* [en línea]. 8 octubre 2014. S.l.: s.n. Disponible desde: http://aprenderaprogramar.com/index.php?option=com_content&view=article&id=57&Itemid=86.

2. MATO GARCÍA, Rosa María. *Informática III*. S.l.: Editorial Universitaria, 2009.

3. HERNÁNDEZ VELOZO, Adrian. *Módulo de resolución automática de conflictos para el Replicador de Datos Reko*. La Habana: Universidad de las Ciencias Informáticas, 2014.

4. PÉREZ TARANCÓN, Jofman. *Solución integrada de réplica de información*. S.l.: Universidad de las Ciencias Informáticas, 2008.

5. BASTO MUGUERCIA, Enrique y CARBONEL HIDALGO, Luis Angel. *Sincronización de datos del proceso de réplica de estructura para el Replicador de datos REKO*. La Habana: Universidad de las Ciencias Informáticas, 2015.

6. MICROSOFT. Información general del modelo de publicación de replicación. En: [en línea]. [Consultado 28 abril 2017]. Disponible desde: https://msdn.microsoft.com/es-es/library/ms152567(v=sql.120).aspx.

7. IBM KNOWLEDGE CENTER. IBM Knowledge Center - ¿Cómo puedo resolver los conflictos de replicación y grabación? En: [en línea]. [Consultado 16 marzo 2017]. Disponible desde: https://www.ibm.com/support/knowledgecenter/ca/SSKTWP_9.0.0/com.ibm.notes900.help.doc/rep_save_conflicts_t.html.

8. URBANO, Randy. Conflict Resolution Concepts and Architecture. En: [en línea]. febrero 2015. [Consultado 16 marzo 2017]. Disponible desde: https://docs.oracle.com/database/121/REPLN/repconflicts.htm#REPLN005.

9. RIVERA HERNÁNDEZ, Helian. *Módulo de gestión de notificaciones de conflictos de réplica en el Replicador Reko*. La Habana: Universidad de las Ciencias Informáticas, 2014.

10. SYMMETRICDS. SymmetricDS. En: [en línea]. [Consultado 28 abril 2017]. Disponible desde: http://www.symmetricds.org/about/overview.

11. SYMMETRICDS. Sync Schema (DDL) Changes. En: [en línea]. [Consultado 4 diciembre 2017]. Disponible desde: http://www.symmetricds.org/docs/how-to/sync-schema-ddl-changes.

12. DBREPLICATOR. Dbreplicator.org. En: [en línea]. [Consultado 28 junio 2017]. Disponible desde: http://dbreplicator.org/.

13. SOFTWARE, H. DBMoto Cloud Edition generalidades. En: . http://www.hitsw.com/localized/spanish/products_services/dbmoto/dbmoto_cloud/DBMoto_Cloud_Edition.html.

14. RODRÍGUEZ SÁNCHEZ, Tamara. *Metodología de desarrollo para la Actividad productiva de la UCI*. 2015. S.l.: Universidad de las Ciencias Informáticas.

15. VISUAL PARADIGM ENTERPRISE. Visual Paradigm. En: [en línea]. [Consultado 25 enero 2017]. Disponible desde: https://www.visual-paradigm.com/.

16. ORACLE. Java Enterprise Edition. En: [en línea]. [Consultado 25 enero 2017]. Disponible desde: http://www.oracle.com/technetwork/java/javaee/overview/index.html.

17. HOLZNER, Steven. *La biblia de Java 2*. S.l.: Anaya Multimedia, 2000. ISBN 978-84-415-1037-1.

18. RUSS MILES, K. H. *Learning UML 2.0*. O'reilly Media. S.l.: s.n., 2006. 9780596555221. ISBN 0-596-55522-9.

19. PARI, J. Spring Tool Suit Instalacion. En: [en línea]. S.l. 12 marzo 2014. Disponible desde: http://www.slideshare.net/juliopari/spring-tool-suite-instalacion.

20. JDBC. En: [en línea]. 12 marzo 2014. Disponible desde: http://www.alegsa.com.ar/.

21. POSTGRESQL. PostgreSQL: Overview. En: [en línea]. Disponible desde: https://www.postgresql.org/.

22. LEYVA JEREZ, Gloria Raquel. *Componente para el envío y recepción de datos del proceso de réplica de estructura para el software de replicación Reko*. La Habana: Universidad de las Ciencias Informáticas, 2013.

23. PRESSMAN, Roger S. *Software engineering: a practitioner's approach*. S.l.: Palgrave Macmillan, 2005.

24. SOMMERVILLE, Ian. *Ingeniería del Software*. S.l.: s.n., 2005. ISBN 84-7829-074-5.

25. BILLY REYNOSO, Carlos. *Introducción a la Arquitectura de Software* [en línea]. marzo 2004. S.l.: Universidad de Buenos Aires. [Consultado 20 abril 2017]. Disponible desde: http://sunshine.prod.uci.cu/gridfs/sunshine/books/Introduccin_a_la_ASW.pdf.

26. ENGINEERS, T. I. O. E. A. E. *IEEE Recommended Practice for Architectural Description of SoftwareIntensive Systems* [en línea]. S.l.: s.n., 2000. [Consultado 25 abril 2017]. ISBN 7381-2519-9. Disponible desde: http://cabibbo.dia.uniroma3.it/ids/altrui/ieee1471.pdf.

27. MICROSOFT DEVELOPER NETWORK. Desarrollo de software basado en componentes. En: [en línea]. 2013. Disponible desde: http://msdn.microsoft.com/es-es/library/bb972268.aspx.

28. ORTÍZ NORIEGA, Dresky y BOULLÓN LÓPEZ, Carlos Javier. *Módulo de Reko para la resolución de conflictos*. La Habana: Universidad de las Ciencias Informáticas, 2010.

29. GUTIÉRREZ, D. *UML Diagramas de Paquetes* [en línea]. 2011. S.l.: s.n. Disponible desde: http://www.codecompiling.net/files/slides/UML_clase_05_UML_paquetes.pdf.

30. Sparx Systems - Tutorial UML 2 - Diagrama de Despliegue. En: [en línea]. [Consultado 6 junio 2017]. Disponible desde: http://www.sparxsystems.com.ar/resources/tutorial/uml2_deploymentdiagram.html.

31. LÓPEZ SANZ, Marcos. *Proceso Unificado: Implementación* [en línea]. 2008. S.l.: s.n. Disponible desde: http://www.kybele.etsii.urjc.es/docencia/IS_LADE/2010-2011/Material/%5BIS-LADE-2010-11%5DTema4g.PUD.Implementacion.pdf.

32. CALLEJA, M. A. Carmen. *Estándares de codificación* [en línea]. S.l.: s.n. [Consultado 5 mayo 2017]. Disponible desde: http://www.cisiad.uned.es/carmen/estilo-codificacion.pdf.

33. GÓMEZ BARYOLO, Oiner. *Solución Informática de Autorización en Entornos Multientidad y Multisistema*. La Habana: Universidad de las Ciencias Informáticas, 2010.

34. PRESSMAN, Roger S. *Estrategia de prueba del software. En Ingenieria del Software, Un Enfoque Práctico.* España: McGraw-Hil, 2002. ISBN 84-481-3214-9.