Universidad de las Ciencias Informáticas

Facultad 4

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

Soporte para la réplica de estructura entre bases datos gestionadas con Microsoft SQL Server y MySQL para el Replicador de datos REKO.

Autor(es): Jennifer Cáceres Vega.

Jose Manuel Blanco Peña.

Tutor(es): Ing. Gloria Raquel Leyva Jerez.

Ing. Liset Martínez Almaguer.

Co-tutor(es): Ing. Maydalis Hernández Pérez.

Ing. Reiman Alfonso Azcuy.

” La Habana, 29 de junio del 2018”

“Año 60 de la Revolución”

"La clave del éxito en los negocios está en detectar hacia dónde va el mundo y llegar ahí primero."

Bill Gates

DEDICATORIA

Jennifer Cáceres Vega:

Le dedico esta tesis primeramente a mi madre por estar siempre a mi lado cuando la necesite, por darme todo su amor incondicional y estar presente en todos los momentos importantes de mi vida.

A mis abuelos por enseñarme valores de los que estoy muy orgullosa hoy en día y darme sus consejos muy importantes para mí.

Jose Manuel Blanco Peña:

Le dedico esta tesis a mi abuela por darme todo su amor y comprensión.

AGRADECIMIENTO

Jennifer Cáceres Vega:

Primero que todo agradecerle a toda mi familia por apoyarme en esta etapa de mi vida tan importante para mí.

A mi pareja por darme todo su amor, motivarme en todo momento y nunca dejarme dar por vencida.

A mis amigos por los buenos momentos que compartimos y los que nos quedan por compartir.

Por último, a mis tutores por tenerme paciencia y darme parte de su tiempo en todo momento que lo necesite.

Gracias a todos.

Jose Manuel Blanco Peña:

Agradecerle a mi familia, a mis amigos y mis tutores por apoyarme todo este tiempo.

DECLARACIÓN JURADA DE AUTORÍA

Declaramos ser autores 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 firmamos la presente a los ____ días del mes de _____ del año 2018.

Firma de los autores:

_____ _____

Jennifer Cáceres Vega. Jose Manuel Blanco Peña.

Firma de las tutoras:

_____ _____

Ing. Gloria Raquel Leyva Jerez. Ing. Liset Martínez Almaguer.

Firma de los co-tutores:

_____ _____

Ing. Maydalis Hernández Pérez. Ing. Reiman Alfonso Azcuy.

RESUMEN

El siguiente trabajo de diploma presenta una solución para el proceso de réplica de estructura entre bases de datos gestionadas con Microsoft SQL Server y MySQL para el Replicador de datos REKO, el cual permite la actualización automática de los cambios introducidos en la estructura de las bases de datos, mediante la ejecución de consultas del Lenguaje de Definición de Datos. El proceso implementado consiste en capturar los cambios ocurridos en la estructura de la base de datos local y realizar la aplicación de estos en la base de datos distribuida. Para su funcionamiento emplea componentes que garantizan la construcción de las acciones que almacenan los cambios, el envío de las acciones utilizando el protocolo JMS y el servidor de mensajería ActiveMQ, y la aplicación de los cambios en los nodos destinos, empleando para ello el dialecto de los gestores.

En la presente investigación se describe la problemática inicial que da comienzo a la extensión del proceso de réplica de estructura en Microsoft SQL Server y MySQL para el software REKO. Se utilizó métodos teóricos y empíricos y se realizó una comparación con sistemas homólogos arrojando como resultado la necesidad de implementar dicho proceso. Se empleó la metodología AUP en su variante UCI para describir el diseño e implementación del proceso apoyándose en los artefactos que propone. Para culminar, se termina la extensión del proceso el cual fue elaborado con herramientas *Open Source* y librerías de clases con licencias gratuitas y la realización de las pruebas correspondientes a las funcionalidades.

Palabras claves: bases de datos, lenguaje de definición de datos, base de datos distribuidas, dialecto, réplica de estructura.

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

En los 60, cuando las computadoras empezaron a desarrollarse, la atención estaba centrada en la resolución de problemas particulares: si era necesario procesar información, se programaba especialmente una aplicación particular que solucionaba la cuestión. El problema radicaba cuando se compartía información y se necesitaba mantener varias copias en diferentes formatos que requerían las aplicaciones existentes. Este procedimiento, inevitablemente creaba redundancia de la información e inconsistencia, pues los formatos que usaban estas aplicaciones no eran compatibles (1).

Con el avance de las Tecnologías de la Información y la Comunicación (TIC) se extendió la necesidad de crear un sistema al cual accedieran todas las aplicaciones de la organización, es decir, se buscaba que todos los sistemas en una organización compartieran un sólo almacén de datos. De esta forma surge el término Sistema de Gestión de Base de Datos (DBMS, por sus siglas en inglés) como un sistema que servía de proveedor de datos a diversas aplicaciones. Pero originalmente la información se almacenaba de manera centralizada (1). Con el tiempo aumentaron las necesidades de compartir datos y recolectar cualquier tipo de información de diferentes Bases de Datos (BD), todo esto produjo ciertos inconvenientes que no era posible solucionarlos o volverlos eficientes. Estos problemas impulsaron la creación de los Sistemas de Base de Datos Distribuidas (SBDD), los cuales hoy en día proveen características indispensables en el manejo de información; es decir, la combinación de las redes de comunicación y las bases de datos (2).

Las bases de datos distribuidas posibilitan que un usuario pueda acceder desde cualquier sitio a los datos en cualquier parte de la red exactamente como si estos fueran accedidos de forma local. Es por ello que una de las características a tener presente en el desarrollo de estos sistemas es el posicionamiento de los datos y el esquema bajo el cual se desea hacer, una alternativa que existe para su realización es la réplica (2).

Un SBDD soporta replicación de datos cuando un fragmento puede ser representado por muchas copias distintas, o réplicas, guardadas en diferentes sitios. Un objeto replicado puede permanecer disponible para su procesamiento mientras esté disponible al menos una copia. La replicación es necesaria debido a que permite que las aplicaciones puedan operar sobre las copias de una BD local en lugar de tener que comunicarse con otras BD distribuidas. Estas ventajas proporcionan un mayor rendimiento y disponibilidad de los datos (3).

En Cuba, precisamente en la Universidad de Ciencias Informáticas (UCI), se localiza el Centro de Consultoría y Desarrollo de Arquitecturas Empresariales (CDAE), el cual posee el software llamado Replicador de datos REKO, como una solución de réplica en ambientes distribuidos. REKO tiene como objetivo brindar una herramienta multiplataforma que le permite al usuario, con el menor esfuerzo, cubrir las necesidades fundamentales de réplica, tales como: sincronización, transferencia de datos entre diversas localizaciones y la centralización de la información en una única localización, protección y recuperación, así como satisfacer otras necesidades relacionadas con la distribución de datos entre los gestores más populares, dígase PostgreSQL, MySQL, Oracle y Microsoft SQL Server. Ha sido utilizado en el Sistema de Gestión Penitenciaria Venezolano y se ha adoptado como solución de réplica en Sistema de Gestión de Gobierno, Prevención del Delito, Planificación de Recursos Empresariales, en el Sistema de Gestión de Hospitales Alas-His, entre otros (4).

El software Replicador de datos REKO desde su versión 4.0 brinda soporte al proceso de réplica de estructura para bases de datos gestionadas con PostgreSQL. Sin embargo, cuando el entorno de réplica es entre bases datos gestionadas con Microsoft SQL Server y MySQL los cambios en el Lenguaje de Definición de Datos (DDL[1], por sus siglas en inglés) efectuados no son reconocidos automáticamente por la herramienta, lo que conlleva a que en la configuración de tablas no se observan las nuevas estructuras. Además, los cambios DDL introducidos generan diferencias de estructuras, debido a que no existe una correspondencia entre los cambios realizados durante la generación de reportes en la BD y las estructuras almacenadas en el sistema REKO en tiempo real. Para solucionar este problema es necesario realizar acciones de forma manual, lo cual incurre en pérdidas de tiempo y mayores esfuerzos por parte de los usuarios.

A pesar de que REKO cuenta con soluciones para el envío, recepción y aplicación de cambios DDL, la homogeneidad se mantiene en BD gestionadas con PostgreSQL y Oracle. Por tanto, en BD gestionadas con Microsoft SQL Server y MySQL no se garantiza que se puedan enviar y aplicar estos nuevos cambios hacia las demás bases de datos distribuidas. Al no actualizar automáticamente los cambios realizados sobre la estructura de la BD provoca que aumente el tiempo de respuesta y que se introduzcan errores durante el proceso de réplica de datos, lo que genera diferencias de estructuras con respecto al resto de las bases de datos distribuidas que

[1] Data Definition Language se encarga de la modificación de la estructura de los objetos de la base de datos como: la generación de nuevas tablas, la creación de nuevos campos en tablas ya existentes, el cambio del tipo de dato de un campo, la eliminación de una tabla y la generación de nuevas relaciones.

pueden derivar en problemas de integridad referencial, inconsistencias y tablas o datos incompletos.

REKO cuenta además con el módulo Monitor de réplica que permite observar todas las acciones del proceso de réplica de datos. Sin embargo, el administrador de réplica no puede visualizar el comportamiento del proceso de réplica de estructura, lo que puede provocar pérdida de información durante eventualidades tales como: fallos en la red o la interrupción de alguna instancia. Se dificulta el trabajo del personal encargado de administrar el sistema, debido a que tienen que invertir tiempo y esfuerzo en verificar de forma manual la integridad de los datos replicados e identificar con exactitud el origen del problema.

Por la situación antes expuesta, se identifica el siguiente **problema a resolver**:

¿Cómo garantizar la coherencia en los datos del proceso de réplica con Microsoft SQL Server y MySQL en el Replicador de datos REKO?

Este problema enmarca el **objeto de estudio** referente al proceso de réplica de estructura en sistemas de bases de datos distribuidas.

Para resolver el problema identificado se propone el siguiente **objetivo general:**

Desarrollar un proceso para capturar, aplicar y monitorizar cambios DDL en bases de datos gestionadas con Microsoft SQL Server y MySQL, que permita mantener la coherencia de los datos que intervienen en el proceso de réplica de estructura del Replicador de datos REKO.

Por lo que el **campo de acción** comprende el proceso de captura, aplicación y monitorización de cambios de estructura en BD Microsoft SQL Server y MySQL en replicadores.

Para dar cumplimiento al objetivo general anteriormente planteado se definen las siguientes **tareas de investigación**:

1. Elaboración del marco teórico de la investigación sobre los principales conceptos asociados con la problemática a resolver.
2. Análisis del entorno y dominio del Replicador de datos REKO.
3. Definición de las herramientas a utilizar para la implementación de la aplicación.
4. 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.

5. Diseño e implementación de la propuesta informática para solucionar el problema planteado.
6. Validación de la aplicación.

En la confección de la investigación se utilizaron **métodos científicos**, con el fin de resolver y dar cumplimiento al objetivo.

Métodos teóricos:

- **Histórico - Lógico**: se usó con el objetivo de realizar un estudio de los referentes teóricos, metodológicos, tecnológicos y cronológicos sobre las herramientas de réplica que realizan trabajos similares, donde se obtuvieron conocimientos que fueron aplicados en la solución del problema presentado.
- **Analítico – Sintético**: permitió profundizar en el conocimiento y estudio de los antecedentes y aportó importantes elementos para llegar a la elaboración del proceso de réplica de estructuras en los gestores de BD Microsoft SQL Server y MySQL con el Replicador de datos REKO.

Métodos empíricos:

- **Observación científica**: permitió valorar el estado del problema, recopilar información al respecto, detectar las necesidades existentes y posibles mejoras a incorporar al software, tomando como objetivo fundamental la capacitación y el análisis de elementos que puedan ayudar en el desarrollo de la solución.

El presente trabajo de diploma se encuentra estructurado en **tres capítulos:**

El **Capítulo 1. Fundamentación teórica** presenta un análisis del estado del arte sobre replicadores de datos y los conceptos asociados a la investigación. Destaca las tendencias y tecnologías actuales sobre las que se apoya la propuesta del sistema informático. Describe la metodología de desarrollo de software, herramientas y lenguajes a emplear en el desarrollo de la solución.

El **Capítulo 2. Propuesta de solución** incluye la descripción, diseño y análisis de la solución que se propone para darle respuesta a la problemática planteada. Especifica los requisitos funcionales y los no funcionales, así como las descripciones de los requisitos, estilo arquitectónico, patrones de diseño aplicados, los diagramas de clases del diseño y el modelo de despliegue.

El **Capítulo 3. Implementación y validación** muestra la implementación del proceso. El diseño de los diagramas de componentes y brinda una solución a los requisitos especificados. Muestra el código fuente de las principales clases y la aplicación de pruebas de diferentes tipos al proceso para demostrar el cumplimiento de los requisitos.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

El presente capítulo refleja los principales conceptos asociados con la problemática a resolver, haciéndose necesario realizar un estudio de soluciones similares existentes y de las tecnologías idóneas para la implementación de la propuesta de solución, fundamentando de este modo las bases de la investigación.

1.1 Marco conceptual

En esta sección se detallan los modelos teóricos, conceptos, argumentos e ideas que intervienen en el desarrollo de la presente investigación. Se describe el estado del arte, es decir, señalar las principales líneas teóricas en relación con el tema, de modo que se pueda proponer una mirada teórica en relación con la investigación.

1.1.1 Base de datos

Las bases de datos se desarrollaron a partir de la necesidad de almacenar los datos inicialmente en sistemas centralizados. Sobre todo, desde la aparición de las primeras computadoras, el concepto de bases de datos ha estado siempre ligado a la informática.

La autora Mercedes Marqués[2] plantea:

"Una base de datos es un conjunto de datos almacenados en memoria externa que están organizados mediante una estructura de datos" (5).

Según la Dra. María del Carmen Gómez[3] Fuentes:

"Una base de datos no es más que un conjunto de información (un conjunto de datos) relacionada que se encuentra agrupada o estructurada" (6).

Los autores[4] Rafael Camps Paré, Luis Alberto Casillas Santillán, Dolors Costal Costa, Marc Gibert Ginestà, Carme Martín Escofet y Oscar Pérez Mora exponen:

"Una base de datos de un SI[5] es la representación integrada de los conjuntos de entidades instancia correspondientes a las diferentes entidades tipo del SI y de sus interrelaciones. Esta

[2] Mercedes Marqués pertenece al Departamento de Ingeniería y Ciencia de la Computación de la Universidad Jaume I de Castelló.
[3] Dra. María del Carmen Gómez pertenece al Departamento de Matemáticas Aplicadas y Sistemas de la Universidad Autónoma Metropolitana Unidad Cuajimalpa.
[4] Autores del libro "Software Libre". Bases de datos.
[5] Sistema Informático es un sistema que permite almacenar y procesar información; es el conjunto de partes interrelacionadas: hardware, software y personal informático.

representación informática (o conjunto estructurado de datos) debe poder ser utilizada de forma compartida por muchos usuarios de distintos tipos" (7).

Los autores[6] Eva Gómez Ballester, Patricio Martínez Barco, Paloma Moreda Pozo, Armando Suárez Cueto, Andrés Montoyo Guijarro y Estela Saquete Boro emiten:
"Una base de datos (BD) es un conjunto de datos estructurados apropiadamente y relacionados entre sí" (8).

Teniendo en cuenta los conceptos anteriores se considera que una base de datos es un conjunto de datos almacenados que se encuentran organizados mediante una estructura y que se interrelacionan. Además, pueda ser compartida y utilizada por varios usuarios de distintos tipos.

1.1.2 Base de datos distribuida

En un inicio se pensó almacenar la información de manera centralizada utilizando un conjunto de herramientas que facilitarán esté tipo de almacenamiento. Pero con el paso del tiempo trajo ciertos inconvenientes que no eran posibles solucionar, lo que condujo a la creación de las bases de datos distribuidas.

Según el autor M.C. Alejandro Gutiérrez Díaz:

"Una Base de Datos Distribuida (BDD) es un conjunto de múltiples bases de datos lógicamente relacionadas las cuales se encuentran distribuidas entre diferentes sitios interconectados por una red de comunicaciones, los cuales tienen la capacidad de procesamiento autónomo lo cual indica que puede realizar operaciones locales o distribuidas" (2).

El autor A. Jaime Elizondo[7] expone:

"Son varias BD interrelacionadas lógicamente y situadas en diferentes nodos de una red de ordenadores" (9).

La autora Martha Eliana[8] expresa:

"Una BDD (Base de Datos Distribuida) es un conjunto de Bases de Datos relacionadas lógicamente, pero que se encuentran físicamente localizadas en varios "sitios" de la red" (10).

[6] Los autores pertenecen al Dpto. de Lenguajes y Sistemas Informáticos de la Escuela Politécnica Superior. Universidad de Alicante.
[7] A. Jaime Elizondo pertenece a la Facultad de ciencias, estudios agroalimentarios e informática. Universidad de la Rioja.
[8] Martha Eliana pertenece a la Universidad del Cauca.

Tomando en cuenta los conceptos anteriormente planteados se ha definido que una base de datos distribuida es un conjunto de bases de datos lógicamente relacionadas, que se encuentran situadas en diferentes sitios interconectadas por una red.

1.1.3 Esquema de una BD

El esquema de una base de datos describe la estructura de una Base de datos, en un lenguaje formal soportado por un Sistema Administrador de Base de datos (DBMS, por sus siglas en inglés). En una Base de Datos Relacional, el esquema define sus tablas, sus campos en cada tabla y las relaciones entre cada campo y cada tabla (11).

Los autores de la presente investigación asumen la definición anteriormente citada, porque está formalmente fundamentada y acorde al objetivo trazado; considerando que un esquema de base de datos define las tablas, los campos de cada tabla, las relaciones entre cada campo y cada tabla; estos son cambios DDL que se pueden introducir y definen la estructura lógica de la base de datos.

1.1.4 Lenguaje de Definición de Datos

El lenguaje de definición de datos se encarga de la modificación de la estructura de los objetos de la base de datos (12).

Los comandos para definir datos son:(13)

CREATE: utilizado para crear nuevas estructuras.

ALTER: utilizado para modificar las estructuras.

DROP: utilizado para eliminar las estructuras.

1.1.5 Lenguaje de Manipulación de Datos

Un lenguaje de manipulación de datos permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado (12).

Los comandos para manipular datos son:(12)

INSERT: es utilizada para agregar uno o más registros a una tabla en un BD relacional.

UPDATE: es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

SELECT: es utilizada para consultar registros de la BD que satisfagan un criterio determinado.

DELETE: es utilizada para borrar uno o más registros existentes en una tabla.

1.1.6 Réplica

La replicación se utiliza para distribuir datos entre diferentes ubicaciones y entre usuarios remotos o móviles mediante redes locales y de área extensa, conexiones de acceso telefónico, conexiones inalámbricas e Internet. Es un recurso utilizado ampliamente en los sistemas distribuidos y que ayuda a los sistemas de datos a mejorar el rendimiento, la disponibilidad y la tolerancia a fallos.

Según el sitio oficial de Microsoft:

"La replicación es un conjunto de tecnologías destinadas a la copia y distribución de datos y objetos de base de datos desde una base de datos a otra, para luego sincronizar ambas bases de datos y mantener su coherencia" (14).

El autor Randy Urbano expresa:

"La replicación es el proceso de compartir objetos y datos de bases de datos en múltiples bases de datos. Para mantener datos y objetos de bases de datos replicados en múltiples bases de datos, se comparte un cambio a uno de estos objetos de base de datos en una base de datos con las otras bases de datos. De esta forma, los datos y los objetos de la base de datos se mantienen sincronizados en todas las bases de datos en el entorno de replicación" (15).

Analizando los conceptos planteados anteriormente se entiende por réplica a un conjunto de tecnologías utilizadas para la copia y distribución de información de una base de datos a otra, para luego sincronizarlas y mantener la coherencia de sus datos.

1.1.7 Coherencia de los datos

Debido a que la información de la base de datos se almacena en forma unificada y coordinada, en todos los tratamientos se utilizan los mismos datos, por lo que los resultados de estos son coherentes (16).

Según el autor Dharmo Rojas[9]:

"La coherencia de los datos estadísticos está referida a la idoneidad de los datos para ser combinados de diferentes maneras y para diferentes usos" (17).

La autora Graciela Echegoyen[10] emite que:

"…es la idoneidad de los datos para ser combinados en forma fiable de diferentes maneras y distintos usos, procedan de una fuente única o investigaciones estadísticas de diversa naturaleza" (18).

En la presente investigación se asumen las definiciones de los autores anteriormente citados y se considera que la coherencia de los datos en el contexto de réplica se entiende como la idoneidad de los datos para realizarse diferentes tratamientos a una base de datos local de forma fiable y después de enviados a otra se conserve la integridad de los datos.

1.1.8 Monitorización

El autor Pedro J. Hernández[11] expone:

"Entendemos por monitorización la medición sistemática y planificada de indicadores de calidad: una actividad que tiene como objetivo identificar la existencia de situaciones problemáticas que hay que evaluar o sobre las que hay que intervenir" (19).

Los autores[12] Xavier Molero, Carlos Juiz y Miguel J. Rodeño expresan:

"La monitorización de un sistema informático permite supervisar, analizar y evaluar el comportamiento del mismo para conocer en qué momentos ocurren ciertas circunstancias que degradan su funcionamiento y, por tanto, afectan a su rendimiento y a su fiabilidad" (20).

El equipo de desarrollo considera utilizar en la investigación la definición de monitorización emitida por los autores Xavier Molero, Carlos Juiz y Miguel J. Rodeño debido a que es la más ajustada a la temática abordada.

[9] Dharmo Rojas autor del libro "Propuesta metodológica para el desarrollo y la elaboración de estadísticas ambientales en países de América Latina y el Caribe".

[10] Graciela Echegoyen autora del libro "Registros administrativos, calidad de los datos y credibilidad pública: presentación y debate de los temas sustantivos de la segunda reunión de la Conferencia Estadística de las Américas de la CEPAL".

[11] Pedro J. Hernández autor del libro "Qué, Cómo y Cuándo Monitorizar: Marco Conceptual y Guía Metodológica".

[12] Autores del libro "Evaluación Y Modelado Del Rendimiento De Los Sistemas Informáticos".

1.2 Replicadores

La replicación sobre BDD está aumentando día a día con respecto a su importancia. Los ambientes modernos de computación demandan soluciones innovadoras y mecanismos flexibles que puedan soportar diferentes formas de replicación, para lograr mejoras de disponibilidad y sincronización en sus sistemas de bases de datos. Algunas contienen la solución de réplica en la aplicación y otras se pueden adquirir de manera independiente. En la presente investigación se realizó un estudio de replicadores para conocer cómo realizan el proceso de captura, aplicación y monitorización de los cambios de estructura, en busca de una propuesta de solución para el proceso de réplica de estructura en los gestores de BD Microsoft SQL Server/ MySQL con el Replicador de datos Reko.

1.2.1 SymmetricDS

SymmetricDS es un software de replicación asincrónica[13], soporta múltiples suscripciones y de sincronización bidireccional, usa tecnologías web y de bases de datos para replicar tablas entre bases de datos relacionales. Está desarrollado en Java lo que le proporciona su carácter multiplataforma, es catalogado como software libre y de código abierto lo que permite una alta capacidad de incremento de nuevas funcionalidades, así como la corrección de errores (21).

Fue diseñado para escalar a un gran número de BD, trabajar con conexiones de bajo ancho de banda, resistir a espacios de tiempo de inoperatividad de la red y se comporta como múltiples-maestros. Además, funciona con gestores de BD: MySQL, Oracle, Microsoft SQL Server, PostgreSQL, DB2, Firebird, HSQLDB, H2, y Apache Derby. La aplicación SymmetricDS permite que los cambios de entrada y de salidas sean sincronizadas con otras BD (22).

Las principales características de SymmetricDS son:

- Licencia de código abierto (Licencia Pública General, GPL[14] por sus siglas en inglés), multiplataforma.
- Pensado para tener nodos desconectados por largos periodos de tiempo.
- Es capaz de replicar los cambios estructurales de la BD que puedan surgir.

[13] Tecnología de replicación que proporciona tolerancia a fallos en servidores y almacenamiento en red. Trabaja capturando los cambios en los ficheros en el nivel del sistema operativo. Una vez que los datos han sido escritos en el sitio de almacenamiento primario, nuevas escrituras a ese sitio pueden ser aceptadas, sin tener que esperar que el sitio de almacenamiento secundario o remoto también termine su escritura.

14 El software que se publica debe ser software libre. Para que sea libre hay que publicarlo bajo una licencia de software libre. Generalmente se utiliza la Licencia Pública General (GPL).

- Flexibilidad a la hora de declarar las reglas de replicación de los datos.
- Facilidad de configuración, basado en *triggers*[15].
- Amplia documentación, comunidad, soporte y ejemplos.
- Alto rendimiento en ambientes con problemas de conexión (23).

SymmetricDS a pesar de contar con grandes potencialidades para ser desplegado en diferentes ambientes de replicación resulta un poco engorrosa la administración y control, debido a que no cuenta con una interfaz que posibilite la administración visual de todos los objetos y componentes que contiene el mecanismo.

Interfaz web para la administración y monitorización para la herramienta de réplica de datos SymmetricDS.

Es una aplicación web desarrollada en la Facultad de Ciencias y Tecnologías Computacionales de la UCI. El lenguaje utilizado para su implementación fue Java con el framework Spring Java. Permite la configuración y monitorización de réplicas con SymmetricDS. Tiene como requerimiento inicial, que mediante el terminal se debe configurar los nodos en cada uno de los servidores. Está desarrollada para la versión 3.7.12 de SymmetricDS la cual no es una versión estable (22).

1.2.2 DBMoto

DBMoto es una herramienta tecnológica que provee la replicación y transformación de datos entre bases de datos. Entre las plataformas de bases de datos la rápida circulación de información permite tener un mejor desempeño de las aplicaciones, en ese sentido DBMoto hace todo esto y mucho más (24).

DBMoto Enterprise Manager incluye asistentes gráficos que ayudan a identificar y conectar las bases de datos relacionales de origen y destino, crear tablas y establecer procesos de replicación customizados[16]. Las reglas y la replicación de datos se mantienen en un metadato, que se puede almacenar en cualquier base de datos (21).

DBMoto fue especialmente diseñado para replicar datos en modo:(24)

[15] Desencadenadores o TRIGGERS. Un desencadenador (o Trigger) es una clase especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento en el servidor de bases de datos.
[16] Producto o servicio que es diseñado o adaptado a las necesidades específicas y deseos de cada cliente.

- ***Refresh* (actualizar):** lee todos los datos cumpliendo las reglas definidas en el administrador para luego escribir los resultados en las bases de datos de destino.
- ***Mirroring* (reflejar):** se realiza una replicación de datos en tiempo real basado en los log/journal[17].
- ***Synchronization* (Sincronizar):** provee una poderosa solución, la cual permite replicar datos en tiempo real y además en forma bi-direccional entre las bases de datos de origen y destino.

La intuitiva interface de DBMoto hace muy sencilla su instalación y ofrece un completo control ilimitado sobre las replicaciones y transformaciones utilizando tecnología de scripting[18], libre de programación (24).

Algunas de sus características son:

- Soporta las principales bases de datos: IBM DB2 UDB (Incluidos los sistemas i/AS400 y z/OS), IBM Informix, IBM Netezza, Oracle, Microsoft SQL Server, MySQL, Sybase Adaptive Server Enterprise, SQL Anywhere, Ingres, Gupta Technologies SQL Base, EnterpriseDB, PostgreSQL, Firebird y MS Access.
- Rápido y fácil ambiente de configuración y administración en ambiente Windows.
- No requiere programación en las plataformas de bases de datos de origen y destino.
- Completa customización vía Microsoft VB.NET scripting orientado a eventos.
- Transformación de funciones built-in, con funcionalidad customizable en cualquier momento.
- Acceso remoto para administración y configuración.
- Completo log de reportes y accesibilidad.
- Poderosa herramienta visual con información sobre el estatus de la replicación.
- Acceso a datos orientado a objetos sobre ambas bases de datos origen y destino.
- Creación automática de tablas de destino.
- Protocolos estándares (.NET, OLE DB y ODBC) para acceso a las tablas de origen y destino (24).

1.2.3 Slony I

[17] Mecanismo por el cual un sistema informático puede implementar transacciones.

[18] Son un conjunto de instrucciones generalmente almacenadas en un archivo de texto que deben ser interpretados línea a línea en tiempo real para su ejecución; esto los distingue de los programas (compilados), pues estos deben ser convertidos a un archivo binario ejecutable.

Slony I es un mecanismo de replicación desarrollado específicamente para PostgreSQL. Es asincrónico, su replicación es lógica y está desarrollado en C y se integra perfectamente al gestor PostgreSQL. Presenta un sistema de replicación que puede configurarse para ser de cascada, es decir un servidor cliente de la replicación puede ser a su vez proveedor de datos para otro servidor cliente. Permitiendo manejar la carga de replicación en los servidores. Es un sistema rápido y que consume pocos recursos en el servidor que se ejecuta (25).

Slony I realiza las actualizaciones mediante *triggers*, lo que significa que no puede propagar cambios de esquemas y operaciones con objetos. Además, no permite replicar automáticamente cambios en los objetos largos y los cambios hechos por comandos DDL. Dificulta la instalación y configuración debido a la carencia de una interfaz (25).

1.2.4 Oracle Streams

Oracle Streams permite compartir información y cada unidad de información compartida se denomina mensaje, y puede compartir estos mensajes en una transmisión. La transmisión puede propagar información dentro de una base de datos o de una base de datos a otra. Además, enruta la información especificada a los destinos definidos y el resultado es una función que proporciona una mayor funcionalidad y flexibilidad que las soluciones tradicionales para capturar y administrar mensajes, y compartir los mensajes con otras bases de datos y aplicaciones (26).

Al usar Oracle Streams puede capturar, organizar y gestionar mensajes en la base de datos automáticamente, incluidos, entre otros, cambios en el lenguaje de manipulación de datos (DML) y cambios en el lenguaje de definición de datos (DDL). También permite colocar mensajes definidos por el usuario en una secuencia, y puede propagar la información a otras bases de datos o aplicaciones automáticamente. Cuando los mensajes llegan a un destino, puede consumirlos según sus especificaciones (26).

Para realizar la réplica se basa en tres componentes:

- **Captura**: los cambios por acciones DDL o DML son capturados del registro de rehacer y luego son empaquetados en LCR[19]. Los datos y los eventos pueden ser cambiados o formateados por un conjunto predefinido de reglas antes de ser empaquetados en un LCR.

[19] Unidad básica de Captura de Cambios, del inglés Logical Change Records.

- **Puesta en escena**: los LCR se almacenan en el entorno de ensayo hasta que un abonado los recoge para ser utilizado o consumido. El abonado puede ser otro entorno de ensayo o una aplicación de usuario.
- **Consumo**: durante el consumo, los LCR se recogen y se aplican en una base de datos. Se modifican los LCR antes de ser aplicados en la base de datos (26).

Las transformaciones basadas en reglas declarativas cubren un conjunto de escenarios de transformación comunes para LCR de fila, incluyendo el cambio de nombre de un esquema, columna o tabla, adición y eliminación de una columna. Oracle Streams realiza internamente transformaciones declarativas, sin invocar PL/SQL[20] (26).

Una transformación personalizada basada en reglas necesita una función PL/SQL definida por el usuario para realizar la transformación. Oracle Streams invoca la función PL/SQL para realizar la transformación. Una transformación personalizada basada en reglas puede modificar LCR o mensajes de usuario. Por ejemplo, una transformación personalizada basada en reglas puede cambiar el tipo de datos de una columna en particular en un LCR (26).

1.2.5 Microsoft SQL Server Developer Network (MSDN)

Microsoft SQL Server Developer Network se basa en tres tipos de replicación los cuales le proporciona un sistema potente y flexible para sincronizar datos en toda su empresa. La replicación a SQLCE 3.5 y SQLCE 4.0 es compatible con Windows Server 2012 y Windows 8 (14).

Los tipos de replicación son:

- **Transaccional**: generalmente se usa en escenarios de servidor a servidor que requieren un alto rendimiento, que incluye: mejorar la escalabilidad y la disponibilidad; almacenamiento de datos e informes; integrando datos de múltiples sitios; integrando datos heterogéneos; y descargando el procesamiento por lotes.
- **Merge**: está diseñada principalmente para aplicaciones móviles o aplicaciones de servidores distribuidos que tienen posibles conflictos de datos. Los escenarios comunes incluyen: intercambiar datos con usuarios móviles; aplicaciones de punto de venta al consumidor (POS); e integración de datos de múltiples sitios.

[20] Lenguaje de Procedimientos/ Lenguaje de Consulta Estructurado, del inglés *Procedural Language/Structured Query Language.*

- **Instantáneas**: se usa para proporcionar el conjunto de datos inicial para la replicación transaccional y de fusión; también se puede usar cuando las actualizaciones completas de datos sean apropiadas (14).

Como alternativa a la replicación, puede sincronizar bases de datos utilizando Microsoft Sync Framework. Sync Framework incluye componentes y una API[21] intuitiva y flexible que facilita la sincronización entre las bases de datos de SQL Server, SQL Server Express, SQL Server Compact y SQL Azure. Sync Framework también incluye clases que se pueden adaptar para sincronizar entre una base de datos de SQL Server y cualquier otra base de datos que sea compatible con ADO.NET (14).

1.2.6 Replicador de datos REKO

El Replicador de datos REKO es una solución de réplica en bases de datos distribuidas. Fue diseñado e implementado completamente usando herramientas *Open Source* y librerías de clases con licencia gratuita. Permite que dos BD puedan replicar entre sí aunque se encuentren en subredes distintas y utilicen protocolos de transmisión diferentes, la única restricción es que todos los nodos deben estar conectados a un servidor central JMS[22] o en caso de un clúster[23] de servidores JMS. Con el empleo de herramientas libres y la metodología de desarrollo AUP[24] (27).

Su desarrollo sobre la plataforma JEE[25] permite que pueda ser ejecutado en cualquier sistema operativo. Se utiliza un diseño desacoplado que ha permitido extender con poco esfuerzo sus funcionalidades. Se emplea, además, el *framework*[26] Spring. Ha sido diseñado para permitir la réplica y sincronización de datos con conexión y sin conexión, la selección de los datos a replicar y la definición de filtros de replicación, replica datos entre bases de datos con estructuras heterogéneas, y entre gestores heterogéneos (27).

Permite la creación de configuraciones para las acciones DDL donde se define la captura de los cambios de estructuras sobre toda la información. Cuando en la BD se genera cualquier cambio de estructura sobre la acción que es especificada en la configuración, todos los cambios son

[21] Application Programming Interface, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
[22] Servicio de Mensajería de Java, del inglés *Java Message Service.*
[23] Grupo de múltiples ordenadores_unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio.
[24] Proceso Unificado Ágil.
[25] Plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java, del inglés *Java Enterprise Edition.*
[26] Arquitectura reusable que brinda la estructura genérica y de comportamiento para un grupo de abstracciones de software.

replicados, por lo que los cambios no deseados por el cliente son enviados hacia las instancias que intervienen en el proceso de réplica de estructura (27).

Principales funcionalidades que ofrece el software Reko en su versión 4.0:

- Soporta la réplica de acciones a través de *triggers*.
- Soporte para réplica de datos en ambientes con conexión y sin conexión.
- Soporte para réplica de datos entre los gestores PostgreSQL, Oracle, MySQL y Microsoft SQL Server.
- Soporte para réplica entre bases de datos con diferentes estructuras.
- Monitoreo en tiempo real de los datos de réplica.
- Réplica de estructura programada para PostgreSQL, se utiliza para iniciar la captura y envío de los cambios de esquemas.
- Seguridad de los datos que se envían con encriptación SSL[27].
- Interfaz de administración de réplica (27).

1.2.7 Resultados de la investigación

Al finalizar la investigación anterior se realizó una comparación entre las herramientas como se muestra en la Tabla 1 la cual concluyó en que las soluciones de réplicas investigadas anteriormente no cumplen con los requerimientos básicos trazados puesto que:

- SymmetricDS a pesar de contar con grandes potencialidades para ser desplegado en diferentes ambientes de replicación, la versión utilizada para el desarrollo de la herramienta administrativa no es estable por lo que se puede decir que esta herramienta no es una solución fiable para el proyecto.
- Las soluciones libres por sí solas no poseen un módulo que se pueda integrar al Replicador de datos REKO y que resuelva los problemas que se generan al realizar cambios a la estructura de las BD gestionadas por Microsoft SQL Server y MySQL.
- Slony I no permite replicar automáticamente cambios en los objetos largos y los cambios hechos por comandos DDL. Además, dificulta la instalación y configuración debido a la carencia de una interfaz y solo permite la réplica con el gestor de base de datos PostgreSQL.
- En algunos casos se hace referencia a la realización de los procesos, pero no ofrecen detalles en cuanto a su funcionamiento interno o de cómo se realiza.

[27] Capa de conexión segura, del inglés *Secure Sockets Layer.*

- Las soluciones privativas no son una opción ya que se debe invertir grandes sumas de dinero en pago de licencias del software, lo cual incurriría en pérdidas para el país.

Tabla 1 Características de los replicadores

Especificaciones/ Herramientas	SymmetricDS	DBMoto	Slony I	Oracle Streams	MSDN	Replicador de datos REKO
Dirección de transmisión de datos						
Maestro-esclavo			X			
Multi-maestro	X	X		X	X	X
Gestores soportados						
PostgreSQL	X	X	X			X
MySQL	X	X				X
Oracle	X	X		X	X	X
Microsoft SQL Server	X	X		X	X	X
Otros gestores	X	X		X	X	
Mecanismos de captura de cambios DDL						
Triggers	X		X		X	X
Logs		X				
Script						
Mensajes				X		
Funciones						
Tipo de licencia						
Libre	X	X	X			X
Privativa				X	X	

Fuente: elaboración propia

Por los aspectos mencionados anteriormente se decidió llevar a cabo la implementación de una solución propia para el Replicador de datos REKO que contribuya a la independización tecnológica del país y de soluciones propietarias.

1.3 Metodología de desarrollo

Las metodologías para el desarrollo imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Tiene como principal objetivo aumentar la calidad del software que se produce en todas y cada una de sus fases de desarrollo, haciendo

énfasis en la calidad y menor tiempo de construcción del software o lo que es lo mismo “producir lo esperado en el tiempo esperado y con el coste esperado” (28).

La Universidad de las Ciencias Informáticas define como política a seguir en los proyectos productivos la utilización de la metodología Proceso Unificado Ágil (AUP, por sus siglas en inglés) 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.

Descripción de la metodología AUP

El Proceso Unificado Ágil fue creado por Scott Ambler, es una versión simplificada del Proceso Unificado de Rational (RUP, por sus siglas en inglés). 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 (29).

Metodología AUP-UCI

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 (30).

Fases de AUP-UCI:

- **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.
- **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 (30).

Escenarios para la disciplina de requisitos:

- **Escenario No 1**: Proyectos que modelen el negocio con Casos de Uso del Negocio (CUN), solo pueden modelar el sistema con Casos de Uso del Sistema (CUS).
- **Escenario No 2**: Proyectos que modelen el negocio con Modelo Conceptual (MC) solo pueden modelar el sistema con CUS.
- **Escenario No 3**: Proyectos que modelen el negocio con Descripción de Proceso de Negocio (DPN), solo pueden modelar el sistema con Descripción de Requisitos por Proceso DRP.
- **Escenario No 4**: Proyectos que no modelen negocio solo pueden modelar el sistema con Historias de usuario (HU) (30).

El proyecto Replicador de datos Reko emplea el escenario No 4 para modelar el sistema a partir de las HU. Este escenario 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 (30).

1.4 Herramientas y tecnologías

En este epígrafe se abordarán los conceptos básicos relacionados con las herramientas que utiliza el Replicador de datos REKO desde versiones anteriores. Las cuales aún siguen empleándose debido a que disminuyen la curvatura de aprendizaje y el tiempo de desarrollo. Teniendo en cuenta que al emplear otras herramientas pueden existir problemas de compatibilidad una vez que se lleve a cabo la integración del proceso para la réplica de estructuras.

Lenguaje Unificado de Modelado 2.0

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener y controlar la información sobre tales

sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. También está pensado para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código, así como generadores de informes. La especificación de UML no define un proceso estándar, pero está pensado para ser útil en un proceso de desarrollo iterativo. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos (31).

Visual Paradigm para UML 8.0

Para el modelado se utilizará Visual Paradigm *for* UML 8.0 por ser una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Agiliza la construcción de aplicaciones con calidad y a un menor coste. Posibilita la generación de bases de datos, transformación de diagramas de Entidad-Relación en tablas de base de datos, así como ingeniería inversa de bases de datos (32).

Java

Es un lenguaje de Programación Orientado a Objetos (POO, por sus siglas en inglés), de código abierto, independiente de la plataforma y robusto. La gestión de memoria y punteros es realizada por el propio lenguaje y no por el programador. Posee mecanismos de seguridad incorporados, los cuales limitan el acceso a recursos de las máquinas donde se ejecuta e incorpora herramientas de documentación. Es multihilos lo que permite dividir el trabajo de un programa en diferentes hilos de ejecución (33).

Java Empresarial 1.7 (JEE[28])

La plataforma Java Enterprise Edition (JEE) es el estándar en el software empresarial impulsado por la comunidad. Es de código abierto, mejora la portabilidad de la aplicación y aumenta la productividad del desarrollador. Java EE permite crear servicios comerciales escalables para aplicaciones web y móviles dinámicas (22).

[28] Java Empresarial, del inglés *Java Enterprise Edition.*

Eclipse

Eclipse es una plataforma de desarrollo *Open Source* basada en Java. Es un desarrollo de IBM cuyo código fuente fue puesto a disposición de los usuarios. En sí mismo Eclipse es un marco y un conjunto de servicios para construir un entorno de desarrollo a partir de componentes conectados (plug-in). Incluye soporte para los lenguajes de programación como C/C++ y COBOL (34).

Apache ActiveMQ 5.9.0

Apache ActiveMQ es el servidor de mensajes de fuente abierta y de patrones de integración más popular y potente. Apache ActiveMQ es rápido y fue diseñado para comunicarse a través de una serie de protocolos con el apoyo de diferentes lenguajes de programación, viene con patrones de integración empresarial fáciles de usar y muchas funciones avanzadas, al tiempo que es totalmente compatible con JMS 1.1 y J2EE 1.4. Apache ActiveMQ se lanza bajo la licencia de Apache 2.0 (35).

Apache Tomcat 7.0

El software Apache Tomcat es una implementación de código abierto de las tecnologías Java Servlet, JavaServer Pages, Java Expression Language y Java WebSocket. Se desarrolla en un entorno abierto y participativo (36).

Microsoft SQL Server 11.0

Microsoft® SQL Server™ es un sistema de administración y análisis de bases de datos relacionales de Microsoft para soluciones de comercio electrónico, línea de negocio y almacenamiento de datos. Proporciona un rendimiento, una disponibilidad y una facilidad de uso innovador para las aplicaciones más importantes. Ofrece capacidad en memoria en la base de datos principal para el procesamiento de transacciones en línea (OLTP) y el almacenamiento de datos. Consta de soluciones de copia de seguridad y de recuperación ante desastres (37).

MySQL 5.5.18

La base de datos MySQL potencia las aplicaciones de Web, comercio electrónico y procesamiento de transacciones en línea (OLTP) más exigentes. Es una base de datos totalmente integrada y segura para transacciones, retrotracción, recuperación de fallas y bloqueo

de nivel de fila. MySQL es una base de datos de código abierto que ofrece la facilidad de uso, escalabilidad y rendimiento (38).

1.5 Consideraciones del capítulo

A partir de la investigación realizada es posible concluir que:

- Para la solución del problema no es factible el uso total o parcial de herramientas de réplica ya desarrolladas para la réplica de estructuras.
- Se definieron las diferentes herramientas y la metodología que se utilizarán en el desarrollo del proceso, las cuales guiarán dicho proceso de desarrollo.

CAPÍTULO 2. PROPUESTA DE SOLUCIÓN

Las características del sistema a desarrollar y su diseño detallado son los elementos que se darán a conocer en el capítulo. Se especifica los requisitos funcionales y no funcionales, se describen las historias de usuarios, los patrones arquitectónicos y de diseño aplicados, los diagramas de clases del diseño y el modelo de despliegue. Además, se utilizará la metodología de desarrollo de software AUP-UCI en su escenario No 4 que guiará el proceso de desarrollo.

2.1 Descripción del proceso a automatizar

El Replicador de datos REKO funciona en un entorno de replicación multimaestro 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 (4).

Por cada configuración, se registran las configuraciones independientes de las tablas que se desean replicar. Cada tabla puede ser configurada para replicar según la acción que se efectúe sobre ella: inserción, actualización y/o eliminación. Además, pueden definirse filtros SQL para determinar por cada acción si se replicará o no el cambio (4).

Cuando se aplican cambios DDL sobre BD gestionadas con Microsoft SQL Server y MySQL los cambios no son reconocidos automáticamente en la configuración de réplica de datos siendo una acción invisible en la configuración de réplica de estructura, por lo que se necesita reiniciar la aplicación. Por tanto, al no actualizar inmediatamente los cambios realizados sobre la configuración de réplica, disminuye el aprovechamiento laboral del personal envuelto en este proceso ya que deben solucionar los problemas de forma manual. Posibilitando que se introduzcan errores en la estructura de la BD, trayendo como consecuencia: inconsistencias entre los datos que se replican, tablas o datos incompletos e incluso problemas de integridad referencial, lo cual afecta la coherencia de los datos.

Para darle solución a estas limitantes, se requiere dar soporte al proceso de réplica de estructura entre BD gestionadas con Microsoft SQL Server y MySQL al Replicador de datos REKO.

2.2 Modelo de dominio

La etapa esencial del análisis o investigación orientada a objetos, es una representación visual de las clases conceptuales individuales u objetos del mundo real en un dominio de interés (39).

Utilizando la notación UML, un modelo de dominio se representa con un conjunto de diagramas de clases en los que no se define ninguna operación. Pueden mostrar:(39)

- Objetos del dominio o clases conceptuales.
- Asociaciones entre las clases conceptuales.
- Atributos de las clases conceptuales.

2.2.1 Diagramas de clases del dominio

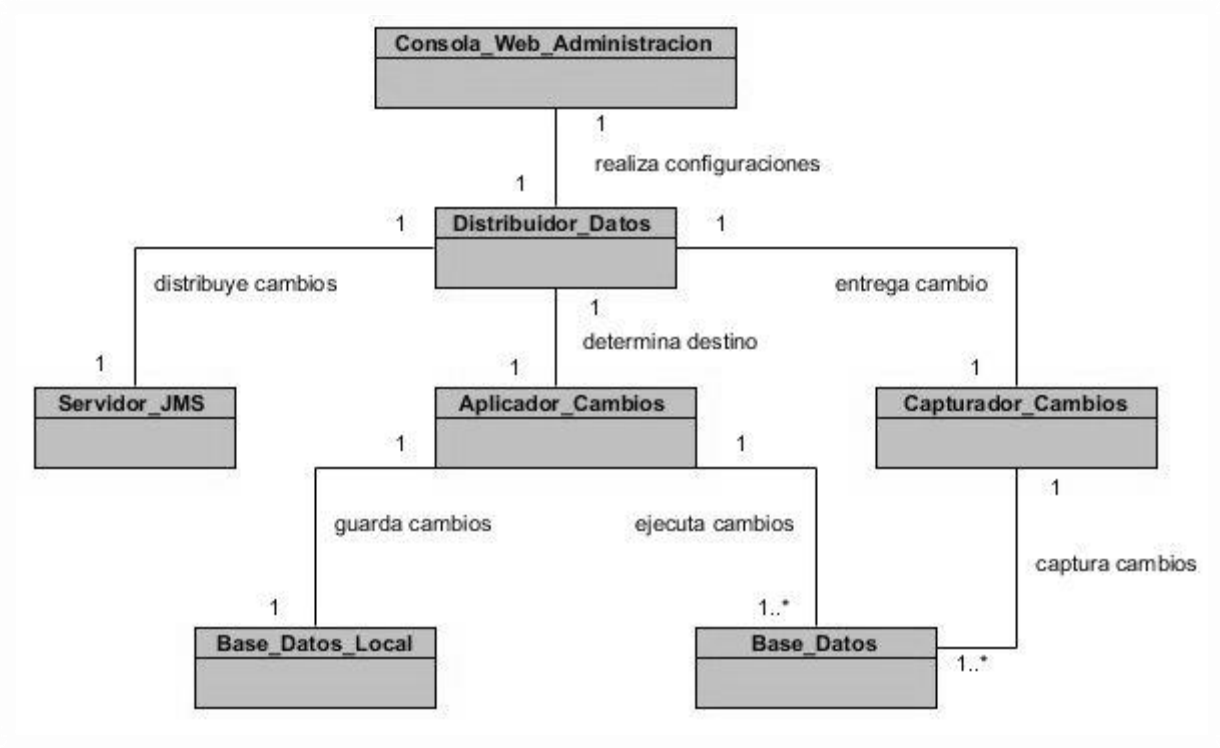


Figura 1 Modelo de dominio

Fuente: elaboración propia.

2.2.2 Descripción de las clases del modelo de dominio

A continuación, para una mejor comprensión del modelo de dominio se describen las clases que lo integran, las cuales son:

Consola_Web_Administración: representa la interfaz del Replicador de datos REKO que permite realizar las configuraciones principales del software como el registro de nodos, configuración de conflictos, configuración de las tablas y esquemas a replicar, y el monitoreo del funcionamiento del sistema.

Distribuidor_Datos: determina el destino de cada configuración de cambio y se responsabiliza de su llegada.

Servidor_JMS: representa el servidor de mensajería, utilizado como punto intermedio para la distribución de la información enviada.

Capturador_Cambios: se encarga de capturar los cambios realizados a la Base_Datos y de entregarlos al Distribuidor_Datos.

Base_Datos: representa la BD que se replica. Los cambios ejecutados sobre ella serán enviados y aplicados otros cambios provenientes de otros nodos de réplica.

Aplicador_Cambios: ejecuta sobre la Base_Datos los cambios que sean replicados hacia esta.

Base_Datos_Local: guarda las configuraciones propias de la réplica, así como acciones sobre la Base_Datos que han provocado conflicto al aplicarse y transacciones que no se aplicaron en su destino.

2.3 Especificación de requisitos de software

Los requisitos de un sistema cumplen un papel importante en el proceso de desarrollo de software, se enfoca fundamentalmente en definir lo que se desea producir. Su principal tarea consiste en la generación de especificaciones correctas que describan con claridad las necesidades de los usuarios o clientes (40).

2.3.1 Requisitos funcionales

Los requisitos funcionales son los que definen las funciones que el sistema será capaz de realizar, describen las transformaciones que el sistema realiza sobre las entradas para producir la salida esperada por los clientes. Estos requerimientos al tiempo que avanza el proyecto de software se convierten en los algoritmos, la lógica y gran parte del código del sistema (40).
Los requisitos funcionales identificados se definen a continuación:

RF1 Crear objetos de la BD para capturar cambios DDL.

RF1.1 Crear *triggers* para capturar cambios DDL en Microsoft SQL Server.

RF1.2 Crear *logs* para capturar cambios DDL en MySQL.

RF2 Capturar los cambios de estructura.

RF3 Guardar en la tabla de control de la BD los cambios capturados.

RF4 Adicionar SQL del lenguaje DDL para aplicar cambios.

RF4.1 Adicionar SQL de Microsoft SQL Server.

RF4.2 Adicionar SQL de MySQL.

RF5 Monitorizar el proceso de réplica de estructura.

2.3.2 Especificación de Requisitos Funcionales

Tabla 2 Especificación de requisitos funcionales

No.	Nombre	Descripción	Prioridad	Complejidad
RF1	Crear objetos de la BD para capturar cambios DDL	Permite crear los triggers para Microsoft SQL Server y los logs para MySQL en la BD.	Alta	Alta
RF2	Capturar los cambios de estructura	Permite que los triggers y logs capturen los cambios de estructura ocurridos en la BD origen, según la configuración de réplica establecida por el usuario.	Alta	Alta
RF3	Guardar en la tabla de control de la BD los cambios capturados	Permite que los triggers y logs guarden en la tabla de control de la BD los cambios de estructura capturados.	Alta	Alta
RF4	Adicionar SQL del lenguaje DDL para aplicar cambios	Permite reconocer la sentencia SQL correspondiente a cada gestor para así reconocer el cambio y poder aplicarlo.	Alta	Media
RF5	Monitorizar el proceso de réplica de estructura	Permite observar el proceso de réplica de estructura, mostrando	Alta	Media

		las acciones que se envían o se reciben.		

Fuente: elaboración propia.

2.3.3 Requisitos no funcionales

Los requisitos no funcionales definen las características que de una u otra forma puedan limitar el sistema (40).

Los requisitos no funcionales del proceso a desarrollar son equivalentes a los del software Replicador de datos REKO al que se integrará. Los cuales se basaron en la ISO 25010 para la taxonomía de los atributos de calidad; se muestran a continuación en la tabla 3:

Tabla 3 Requisitos no funcionales

No.	Atributo de Calidad	Sub-Atributo	Objetivo
RnF1	Fiabilidad	Tolerancia a fallos	Capacidad que presenta el producto para operar según las dificultades que se le presenten de hardware o software.
		Recuperabilidad	Capacidad que presenta el producto para recuperar los datos directamente afectados y restablecer el estado deseado del sistema en caso de interrupción o fallos.
RnF2	Mantenibilidad	Analizabilidad	Capacidad que presenta el producto de facilitar la evaluación del impacto de un determinado cambio sobre el resto del software, diagnosticar las deficiencias o causas de fallos en el software e identificar las partes a modificar.
		Modificabilidad	Capacidad del producto que permite que sea modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.
		Capacidad para ser probado	Capacidad que presenta el producto de facilitar el establecimiento de 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.

RnF3	Adecuación funcional.	Integridad funcional	Capacidad que presenta el producto de que el conjunto de funciones que posee cubra todas las tareas y objetivos del usuario.
		Corrección funcional	Capacidad que presenta el producto de proporcionar los resultados correctos con el grado necesario de precisión.
RnF4	Portabilidad	Adaptabilidad	Capacidad del producto que le permite ser adaptado de forma efectiva y eficiente a diferentes entornos determinados de hardware, software, operacionales o de uso.
RnF5	Seguridad	No repudio	Capacidad que presenta el producto de que las acciones o eventos puedan ser probados y haber tenido lugar, por lo que los eventos o acciones no pueden ser repudiados más tarde.

Fuente: elaboración propia.

Para una mejor comprensión de los requisitos no funcionales y sus atributos de calidad ver Anexos del 1 al 8.

2.4 Propuesta de solución

Para dar solución al objetivo propuesto se utilizarán las funcionalidades del componente Consola Web de Administración para iniciar la configuración de réplica de estructura, dando paso a la creación de las estructuras de control que contendrán información referente a los cambios de estructura por acciones DDL (*create*, *alter* y *drop*) producidas en la estructura de la BD Microsoft SQL Server o MySQL. Para capturar estas informaciones en el caso de Microsoft SQL Server se crearán los *trigge*rs de eventos los cuales se encargan de capturar los cambios en el momento que ocurren y registrarlos en la tabla de control y en MySQL se traducirán los logs que contendrán la información referente a los cambios.

El Capturador de cambios de REKO cada cierto tiempo registra los cambios realizados en una estructura de control (tabla de control[29]) de la BD y crea un objeto de tipo Grupo de Estructura Replicable (RSG, por sus siglas en inglés) que contendrá un grupo de objetos de la clase Acción de Estructura Replicable (RSA, por sus siglas en inglés). El Capturador de cambios construye cada RSA a partir de la información almacenada en la estructura de control y los metadatos de la

[29] Es una tabla de control que se encuentra en la base de datos. Almacena el tipo de acción, nombre del esquema, tabla y columna a replicar, destino del cambio, así como el usuario que ejecuta la acción.

BD al cual accede a través de la clase **MetadataManager** que tiene implementado REKO. Por tanto, la información correspondiente al cambio realizado se almacenará en cada RSA. Finalizado el proceso de captura y construcción de los RSG, el Capturador de cambios es el responsable de entregárselos al Distribuidor de Datos.

El Distribuidor de Datos a partir de estos objetos, las configuraciones registradas y los filtros asociados a las RSA contenidas en los RSG, define los destinos hacia donde se enviará cada RSG, asegurándose además de la propagación de estos utilizando el servidor de JMS ActiveMQ. Enviado cada RSG a su destino, se deberá eliminar las acciones de estructura replicable dentro de la estructura de control. En cada nodo destino, una vez recibido un RSG, pasará al Aplicador de Cambios que ejecutará las acciones contenidas en los RSG en la BD destino gestionadas con Microsoft SQL Server o MySQL.

El módulo de monitoreo provee información sobre el funcionamiento del mecanismo de réplica, el cual muestra las acciones que se envían o se reciben, lo que permite dar un seguimiento en tiempo real al proceso.

2.5 Historias de usuario (HU)

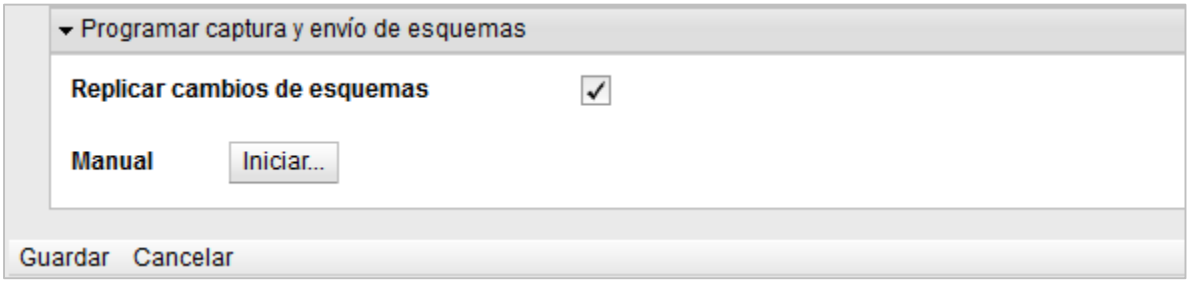
Las Historias de Usuario son una técnica utilizada para especificar los requisitos del software y sirve para que el usuario describa brevemente las características que desea que el sistema sea capaz de realizar. Además, las historias de usuarios son utilizadas por metodologías ágiles y el tratamiento es muy dinámico y flexible, de forma que es lo suficientemente comprensible y delimitado para que los programadores puedan implementarlo y se pueda determinar el costo de dicha implementación (41).

2.5.1 Descripción de las HU

Para un mayor entendimiento del proceso a desarrollar se muestran ejemplos de las HU correspondientes a los requisitos funcionales del sistema. Las restantes HU referentes al proceso están incorporadas desde el Anexo 9 al 11.

Tabla 4 HU1: Crear objetos de la BD para capturar cambios DDL

	Historias de usuario
Número: HU 1	**Nombre del requisito**: Crear objetos de la BD para capturar cambios DDL

Programador: Jennifer Cáceres Vega y Jose Manuel Blanco Peña	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 30 días
Riesgo en Desarrollo: Plan de riesgos	**Tiempo Real**: 120 horas
Descripción: El usuario debe iniciar la configuración de réplica de estructura, una vez iniciado se crean las estructuras de control donde se almacenará la información referente a los cambios de estructura ocurridos en la BD origen por acciones DDL (*create*, *alter* y *drop*), se crean los *triggers* en el caso de Microsoft SQL Server los cuales se encargan de capturar los cambios en el momento que ocurren y registrarlos en la tabla de control y en MySQL se traducirán los *logs* que contendrán la información referente a los cambios. La estructura de control **"reko_structure_control_table"** (tabla de control) debe contener todos los cambios de estructura ocurridos en la BD origen, apoyándose en la información comprendida en las demás estructuras de control.	
Observaciones: La estructuras de control solo se crearán si existe alguna configuración de réplica de estructura establecida y si la opción "Replicar cambios de esquemas" está activada.	
Prototipo de interfaz: 	

Fuente: elaboración propia

2.6 Arquitectura del Replicador de datos REKO

La arquitectura de software es una representación de la estructura del sistema que permite analizar la efectividad del diseño para cumplir los requerimientos establecidos, considerar alternativas arquitectónicas en una etapa en la que hacer cambios al diseño todavía es relativamente fácil y reducir los riesgos asociados con la construcción del software. Además, comprende los componentes del software, sus propiedades externas visibles y las relaciones entre ellos (28).

2.6.1 Estilo y patrones arquitectónicos

El estilo arquitectónico Llamada y retorno es el aplicado en la solución propuesta ya que se recibe una estructura del software que resulta relativamente fácil de modificar y cambiar de tamaño.

Hace especial énfasis en la modificabilidad y escalabilidad; convirtiéndolos en los más generalizados en sistemas a gran escala, en este grupo se encuentran las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, así como también los sistemas orientados a objetos y los jerárquicos en capas (42).

Arquitectura basada en componentes

La arquitectura basada en componentes pone énfasis en la reusabilidad; es decir, en la creación y reutilización de componentes de software. Estos deben catalogarse para facilitar su referencia, estandarizarse para facilitar su aplicación y validarse para facilitar su integración (28).

Las definiciones anteriormente expresadas se pueden emplear para describir en términos generales a la arquitectura del Replicador de datos REKO como basada en componentes, pues sus partes encapsulan un conjunto de comportamientos que pueden ser sustituidos por otros.

El Capturador de cambios, el Distribuidor, el Aplicador y el Administrador son los principales componentes presentes en el software, los cuales serán extendidos para el desarrollo del proceso propuesto. En el paquete **dbstructurechange**, que corresponde al componente Capturador de cambios, se realizarán transformaciones, ya que hasta el momento su funcionamiento consiste en realizar configuraciones de réplica y conflictos de estructuras, este proceso se realiza únicamente para los gestores de BD PostgreSQL y Oracle, actualmente se extenderá su funcionamiento para proveer configuraciones de réplica para los gestores de BD Microsoft SQL Server y MySQL. Con el mismo fin serán modificados los paquetes del Distribuidor, Aplicador y Administrador.

Arquitectura basada en capas

Además de lo planteado anteriormente, el componente Administrador corresponde a un modelo multicapas, donde las capas se muestran como una organización jerárquica tal que cada capa provee servicios a la capa inmediatamente superior y utiliza las prestaciones que le brinda la inmediatamente inferior. Aparte de encontrarse separadas lógica y estructuralmente, las capas se encuentran separadas de manera física (43).

La **capa de presentación** es la que el usuario ve en su ordenador, es donde se tratan los datos que se van a mostrar. Esta capa se comunica únicamente con la capa de negocio (43).

La **capa de negocio** es donde se encuentran los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina como lógica del negocio ya que aquí es donde se establecen todas las reglas que deben cumplirse (43).

En la figura 2 se muestra los principales componentes del proceso de réplica de estructura en el Replicador de datos REKO.

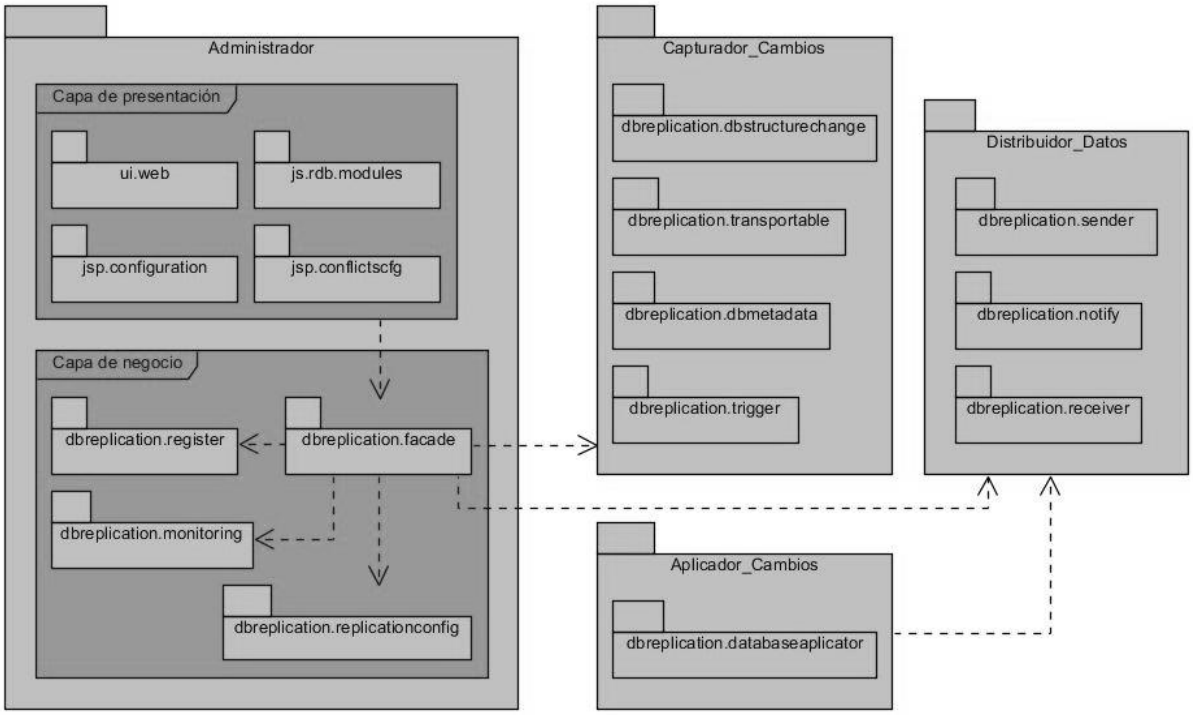


Figura 2 Principales componentes del proceso de réplica de estructura en el Replicador de datos REKO
Fuente: elaboración propia.

2.6.2 Patrones de diseño

Los patrones de diseño describen problemas recurrentes y luego elaboran una solución que puede ser utilizada en otros momentos. Incluyen soluciones basadas en un historial, las cuales están probadas que funcionan. También detallan los módulos, estructuras y mecanismos más profundos del sistema (28).

Patrones *GOF*[30]

Se emplearon patrones GOF que describen soluciones simples y elegantes a problemas específicos en el diseño de software orientado a objetos. Se clasifican en dependencia del propósito para el que han sido definidos: creación, estructurales y de comportamiento (39).

Fachada: Provee una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema. Este patrón es de tipo creación y se utiliza para reducir la dependencia entre clases. Además, ofrece un punto de acceso, de manera que, si las clases son modificadas o se sustituyen por otras, solamente se tiene que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones cliente. REKO cuenta con el paquete *facade* que aprovisiona estas ventajas, la clase "**TransactionProcessor**" ofrece un punto de acceso para iniciar el envío de los *RSG* y el inicio de la aplicación de los cambios en las BD.

Patrones *GRASP*[31]

Los patrones generales de software para la asignación de responsabilidades (GRASP, por sus siglas en inglés) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones (39).

En el desarrollo del proceso se utilizan los cinco patrones de diseño que componen esta clasificación (Controlador, Creador, Alta cohesión, Bajo acoplamiento y Experto) los cuales se evidencian a continuación:

Alta cohesión: se aplica cuando las responsabilidades de una clase están altamente relacionadas con las responsabilidades de otra clase. Esto se evidencia en la relación entre las clases **AplicatorManager** y **DBStructureChangeManager**, ya que **AplicatorManager** utiliza las funcionalidades que brinda la clase **DBStructureChangeManager** por lo que no realizan un trabajo enorme y por tanto pueden ser calificadas como de alta cohesión.

Controlador: es un objeto responsable del manejo de los eventos del sistema, que no pertenece a la interfaz de usuario, el controlador recibe la solicitud del servicio desde la interfaz y coordina su realización delegando a otros objetos. En el proceso desarrollado se utilizan distintas clases controladoras como la **AplicatorManager** y la clase **DBStructureChangeManager.**

[30] Banda de los Cuatro, del inglés *Gang-Of-Four*.
[31] General Responsibility Asignment Software Patterns.

Bajo acoplamiento: este patrón mide el grado en que una clase está conectada a otra, tiene conocimiento de otra, o de alguna manera depende de otra. Es evaluativo, ya que un bajo acoplamiento permite que el diseño de clases sea más independiente, reduce el impacto de los cambios y aumenta la reutilización. Esto se evidencia en las clases **SqlServerDialect** y **MySqlServerDialect** ya que estas se comunican solamente con las clases necesarias para desarrollar cada flujo de evento.

Experto: se basa en asignar la responsabilidad al experto en la información, es decir a la clase que contiene la información necesaria para cumplir la responsabilidad. De esta manera se logra que las clases tengan un mejor comportamiento y hacen que las mismas sean más cohesivas, permitiendo a su vez que sean mayores las posibilidades de soporte. Este patrón se evidencia en la clase **DBStructureChangeManager** que cuenta con la información necesaria para cumplir la responsabilidad de capturar los cambios de estructura y la creación de los *RSG*.

Patrón *DAO*[32]

Además, también se utilizó el patrón DAO para la solución propuesta, el cual divide las responsabilidades de la aplicación en dos aspectos, las clases que se encargaran de la lógica del negocio y las otras de la responsabilidad de persistencia. Ningún tipo de código debe poder acceder al repositorio de datos afuera de las clases DAO. El DAO accede a la fuente de datos y la encapsula para los objetos clientes. Los Objetos de Acceso a Datos pueden usarse en Java para aislar a una aplicación de la tecnología de persistencia Java subyacente, la cual podría ser JDBC (44).

2.7 Modelo de diseño

Los modelos de diseño muestran los objetos o clases de un sistema y donde sea apropiado, los diferentes tipos de relaciones entre estas entidades. Son el puente entre los requerimientos y la implementación del sistema. Por lo que deben ser abstractos con el fin de que el detalle innecesario no oculte las relaciones entre ellos y los requerimientos del sistema. Sin embargo, también tienen que incluir suficiente detalle para que los programadores tomen las decisiones de implementación (45).

2.7.1 Diagramas de paquetes

[32] Objeto de Acceso a Datos, del inglés *Data Access Object*.

Los Diagramas de Paquetes son utilizados para reflejar la organización de los paquetes y sus elementos. Cada paquete corresponde a un submodelo del modelo del sistema que puede contener otros paquetes, sin límite de anidamiento, pero cada elemento pertenece a sólo un paquete. Una clase de un paquete puede presentarse en otro paquete por la importación a través de una relación de dependencia entre paquetes (46).

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

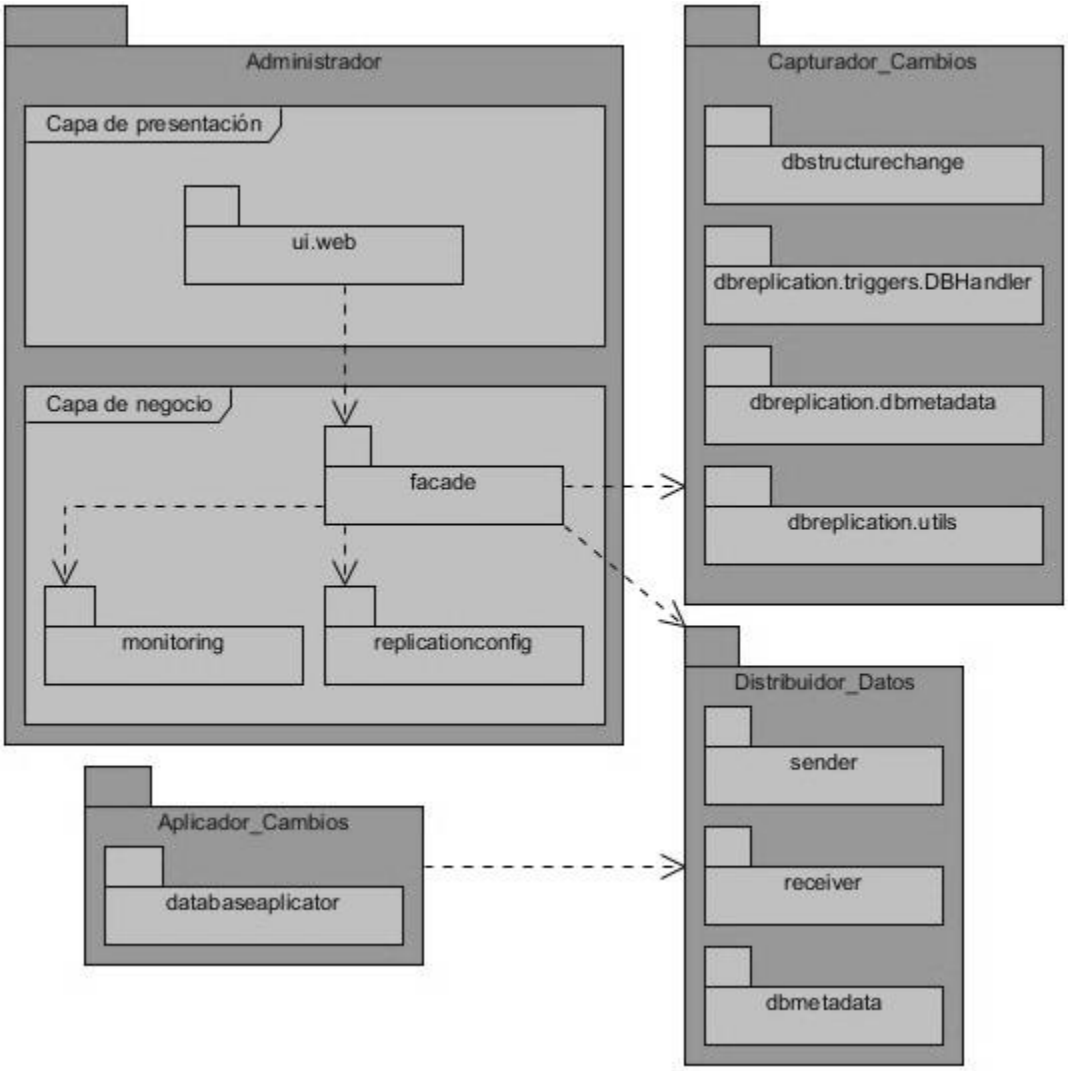


Figura 3 Diagrama de paquetes de la solución

Fuente: elaboración propia.

2.7.2 Diagramas de clases de diseño

Los diagramas de clases describen la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro (47).

A continuación, se muestran los diagramas de clases diseñados para los requisitos funcionales del sistema:

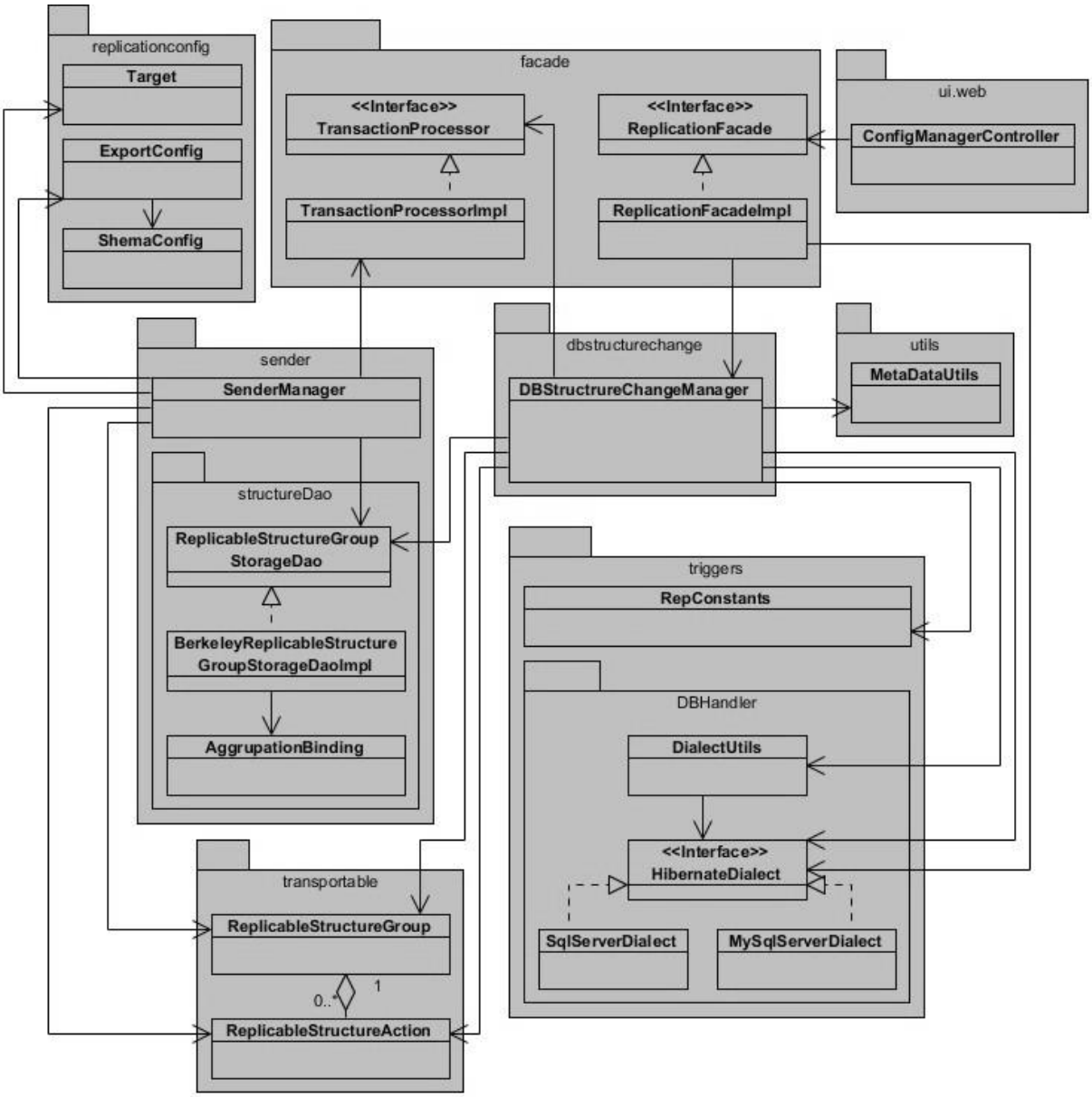


Figura 4 Diagrama de Clases para los RF1 y RF2

Fuente: elaboración propia

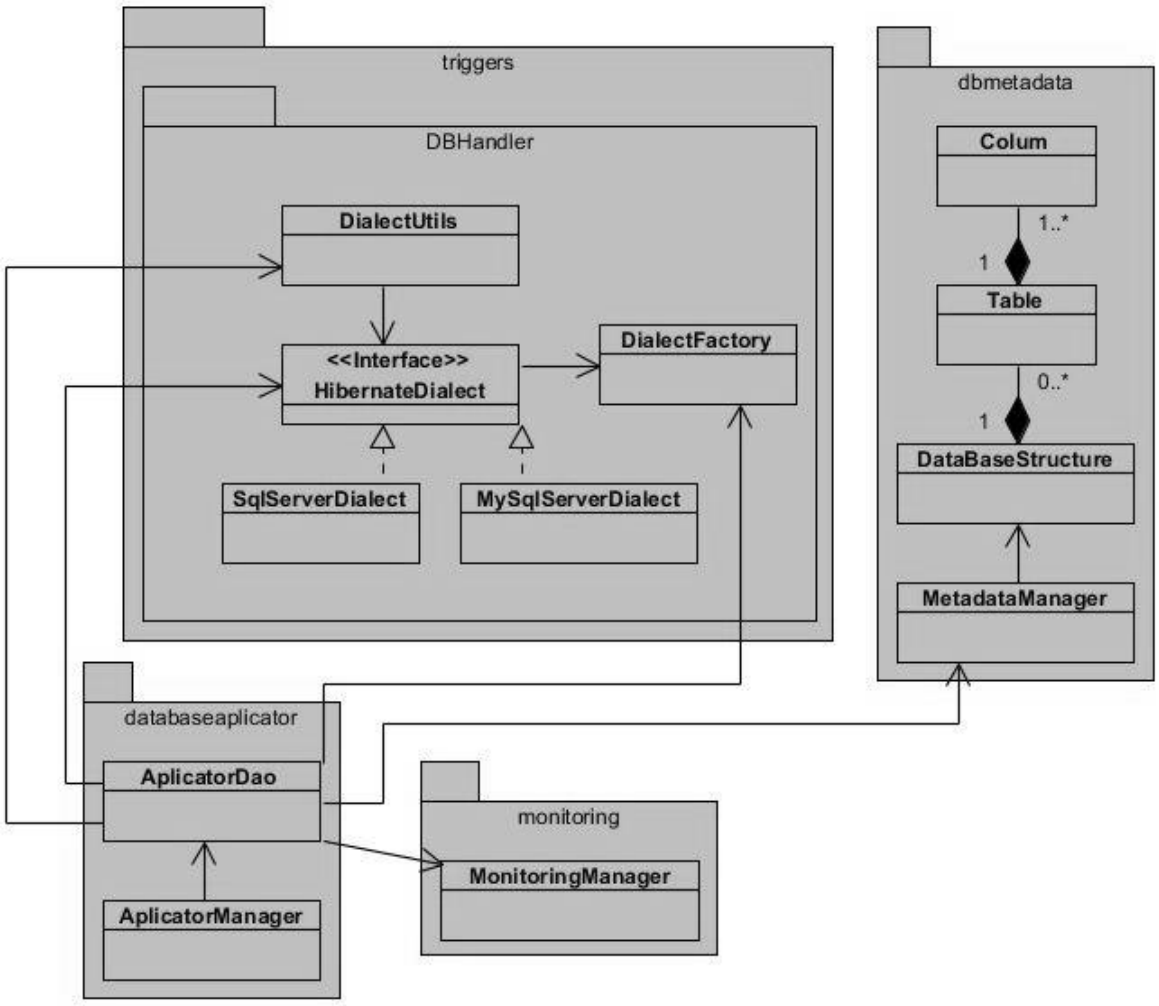


Figura 5 Diagrama de Clases para los RF3, RF4 y RF5

Fuente: elaboración propia

2.7.3 Descripción de las clases del diseño

Tabla 5 Descripción de la clase SqlServerDialect

Descripción de la clase SqlServerDialect	
Nombre: SqlServerDialect	
Tipo de clase: Auxiliar	
Responsabilidades	
Nombre:	createTable(tableName: cu.uci.dbreplication.dbmetadata.Table): String
Descripción:	Obtiene la sentencia sql de crear una tabla.

Nombre:	createSchema(schemaName: String): String
Descripción:	Obtiene la sentencia sql de crear un esquema.
Nombre:	renameTable(oldTableName: String, newTableName: String): String
Descripción:	Obtiene la sentencia sql de renombrar una tabla.
Nombre:	sentenceDropTable(tableName: String): String
Descripción:	Obtiene la sentencia sql de eliminar una tabla.
Nombre:	renameColumn(tableName: String, oldColumName: String, newColumName: String): String
Descripción:	Obtiene la sentencia sql de renombrar una columna.
Nombre:	alterDropColum(tableName: String, columName: String): String
Descripción:	Obtiene la sentencia sql de eliminar una columna
Nombre:	alterDropConstraint(tableName: String, constraintName: String): String
Descripción:	Obtiene la sentencia sql para eliminar una restricción.
Nombre:	dropSchema(schemaName: String): String
Descripción:	Obtiene la sentencia sql para eliminar un usuario o esquema.
Nombre:	alterAddPrimaryKey(table: cu.uci.dbreplication.dbmetadata.Table, pkConstraint: String, columDefinition: String): String
Descripción:	Obtiene la sentencia sql para adicionar una clave primaria.
Nombre:	alterAddUniqueKey(table: cu.uci.dbreplication.dbmetadata.Table,ukConstraint: String, colum: String): String
Descripción:	Obtiene la sentencia sql para adicionar una clave única.
Nombre:	alterAddColumFinish(table: cu.uci.dbreplication.dbmetadata.Table, columNew: cu.uci.dbreplication.dbmetadata.Colum): String
Descripción:	Obtiene la sentencia sql para adicionar una columna.
Nombre:	alterTypeColum (tableName: String, columName: String, newType: String): String
Descripción:	Obtiene la sentencia sql para cambiar el tipo de dato de la columna.
Nombre:	alterDefaultValue(tableName: String, columName: String, newValue: String): String
Descripción:	Obtiene la sentencia sql para agregar valores por defecto a una columna.
Nombre:	alterDropDefault (tableName: String, columName: String): String

Descripción:	Obtiene la sentencia sql para eliminar valores por defecto de una columna.
Nombre:	alterAddForeignKey(table: cu.uci.dbreplication.dbmetadata.Table, fkConstraint: String, colum: cu.uci.dbreplication.dbmetadata.Colum, tableReference: cu.uci.dbreplication.dbmetadata.Table, columReference: cu.uci.dbreplication.dbmetadata.Colum): String
Descripción:	Obtiene la sentencia sql para adicionar una clave ajena.
Nombre:	alterAddCheck(table: cu.uci.dbreplication.dbmetadata.Table, chekConstraint: String, condition: String): String
Descripción:	Obtiene la sentencia sql para adicionar una comprobación.
Nombre:	alterTriggerEnable (tableName: String, triggerName: String): String
Descripción:	Obtiene la sentencia sql para habilitar un trigger.
Nombre:	alterTriggerDisable(tableName: String, triggerName: String): String
Descripción:	Obtiene la sentencia sql para desabilitar un trigger.

Fuente: elaboración propia

Tabla 6 Descripción de la clase MySqlServerDialect

Descripción de la clase MySqlServerDialect	
Nombre: MySqlServerDialect	
Tipo de clase: Auxiliar	
Responsabilidades	
Nombre:	createTable(tableName: cu.uci.dbreplication.dbmetadata.Table): String
Descripción:	Obtiene la sentencia sql de crear una tabla.
Nombre:	createDatabase(databaseName: String): String
Descripción:	Obtiene la sentencia sql de crear una Base de datos.
Nombre:	dropDatabase(databaseName: String): String
Descripción:	Obtiene la sentencia sql de eliminar una Base de datos.
Nombre:	renameTable(oldTableName: String, newTableName: String): String
Descripción:	Obtiene la sentencia sql de renombrar una tabla.
Nombre:	sentenceDropTable(tableName: String): String
Descripción:	Obtiene la sentencia sql de eliminar una tabla.

Nombre:	renameColumn(tableName: String, oldColumName: String, newColumName: String): String
Descripción:	Obtiene la sentencia sql de renombrar una columna.
Nombre:	alterDropColum(tableName: String, columName: String): String
Descripción:	Obtiene la sentencia sql de eliminar una columna
Nombre:	alterDefaultValue(tableName: String, columName: String, newValue: String): String
Descripción:	Obtiene la sentencia sql para agregar valores por defecto a una columna.
Nombre:	alterDropDefault(tableName: String, columName: String): String
Descripción:	Obtiene la sentencia sql para quitar valores por defecto a una columna.
Nombre:	alterSetNotNull(tableName: String, columName: String): String
Descripción:	Obtiene la sentencia sql para agregar el not null a una columna.
Nombre:	alterDropNotNull(tableName: String, columName: String): String
Descripción:	Obtiene la sentencia sql para quitar el not null a una columna
Nombre:	alterAddPrimaryKey(table: cu.uci.dbreplication.dbmetadata.Table, pkConstraint: String, columName: cu.uci.dbreplication.dbmetadata.Colum): String
Descripción:	Obtiene la sentencia sql para adicionar una clave primaria.
Nombre:	alterAddUniqueKey(table: cu.uci.dbreplication.dbmetadata.Table, ukConstraint: String, colum: cu.uci.dbreplication.dbmetadata.Colum): String
Descripción:	Obtiene la sentencia sql para adicionar una clave única.
Nombre:	alterAddColumFinish(table: cu.uci.dbreplication.dbmetadata.Table, colum: cu.uci.dbreplication.dbmetadata.Colum): String
Descripción:	Obtiene la sentencia sql para adicionar una columna.
Nombre:	alterTypeColum (tableName: String, columName: String, newType: String): String
Descripción:	Obtiene la sentencia sql para cambiar el tipo de dato de la columna.
Nombre:	alterAddForeignKey(table: cu.uci.dbreplication.dbmetadata.Table, fkConstraint: String, colum: cu.uci.dbreplication.dbmetadata.Colum, table2: cu.uci.dbreplication.dbmetadata.Table, colum2: cu.uci.dbreplication.dbmetadata.Colum): String

Descripción:	Obtiene la sentencia sql para adicionar una clave ajena.
Nombre:	alterAddCheck(table: cu.uci.dbreplication.dbmetadata.Table, chekConstraint: String, chekConstraint: String): String
Descripción:	Obtiene la sentencia sql para adicionar una comprobación.

Fuente: elaboración propia

Se realizaron otras descripciones de clases donde se utilizó métodos ya implementados y en otros casos se efectuaron nuevas implementaciones de operaciones (ver Anexos del 12 al 16).

2.8 Modelo de despliegue

Es un tipo de diagrama del Lenguaje Unificado de Modelado que se utiliza para modelar el hardware utilizado en las implementaciones del sistemas y las relaciones entre sus componentes (47).

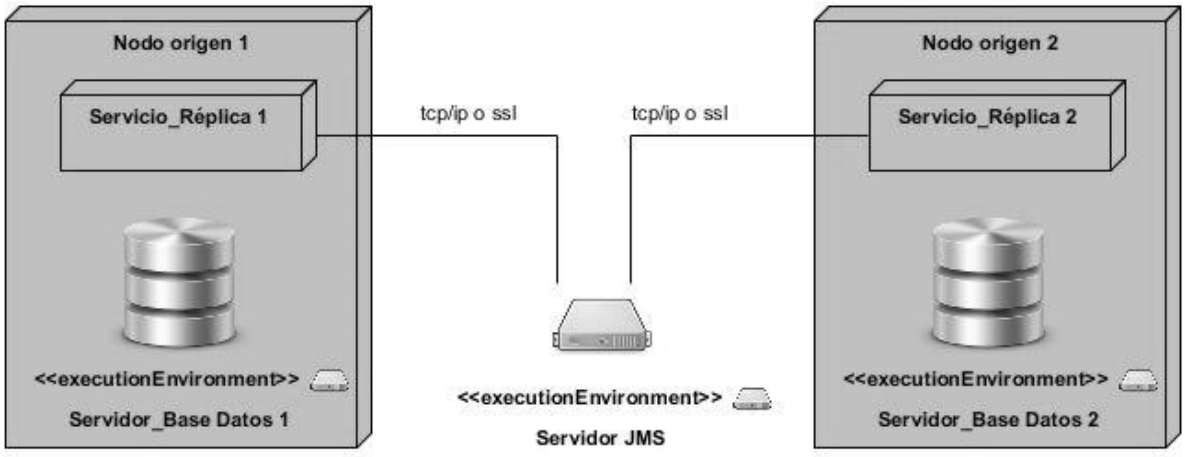


Figura 6 Diagrama de despliegue

Fuente: elaboración propia

2.9 Consideraciones del capítulo

Una vez realizado el análisis y diseño del sistema se obtuvieron las siguientes consideraciones:

- Se hace necesario extender las funcionalidades de los componentes Capturador, Distribuidor y Aplicador de cambios, pues hasta el momento su funcionamiento se basa en capturar los cambios de consultas DDL introducidos solo por los gestores PostgreSQL y Oracle, por lo cual se extenderá su funcionamiento para permitir la actualización

automática de los cambios realizados en la BD por la ejecución de consultas DDL en los gestores Microsoft SQL Server y MySQL.

- La implementación del proceso a desarrollar estará conformada por tres partes esenciales: la captura, aplicación y monitorización de los cambios de estructuras en las BD gestionadas con Microsoft SQL Server y MySQL.

CAPITULO 3. IMPLEMENTACIÓN Y VALIDACIÓN

En el presente capítulo se abordan los aspectos más importantes referentes al proceso de implementación. Se muestra el diagrama de componente, el código fuente de una de las principales clases y por último se realizan pruebas al sistema implementado, a través de los casos de pruebas, para demostrar su validez.

3.1 Modelo de implementación

El modelo de implementación identifica los componentes físicos de la implementación para que puedan comprenderse y gestionarse mejor. Puede incluir directorios y archivos, incluyendo código fuente, datos y archivos ejecutables (39).

3.1.1 Diagramas de componentes

Los diagramas de componentes representan cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes, los cuales pueden ser: archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes (47).

Para una mejor comprensión de la composición física de la implementación del proceso, se muestran los diagramas de componentes agrupados por los sub-sistemas de implementación. El restante diagrama de componentes de la solución se puede ver en el Anexo 17.

Los sub-sistemas de implementación siguientes se relacionan a través de la interfaz **TransactionProcessor** (ver Anexo 18).

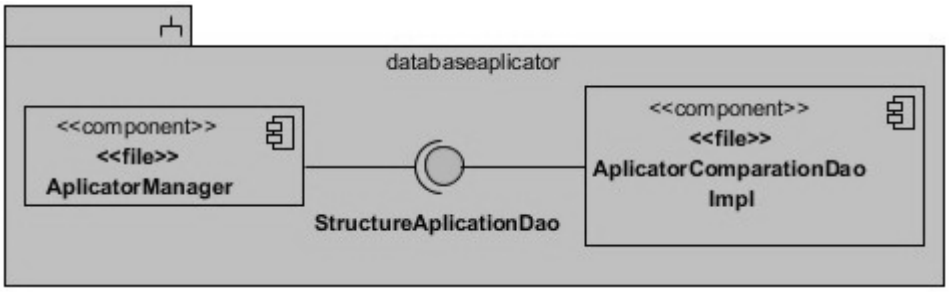


Figura 7 Diagrama de componentes para el subsistema Aplicador de cambios del Replicador de datos REKO. Fuente: elaboración propia.

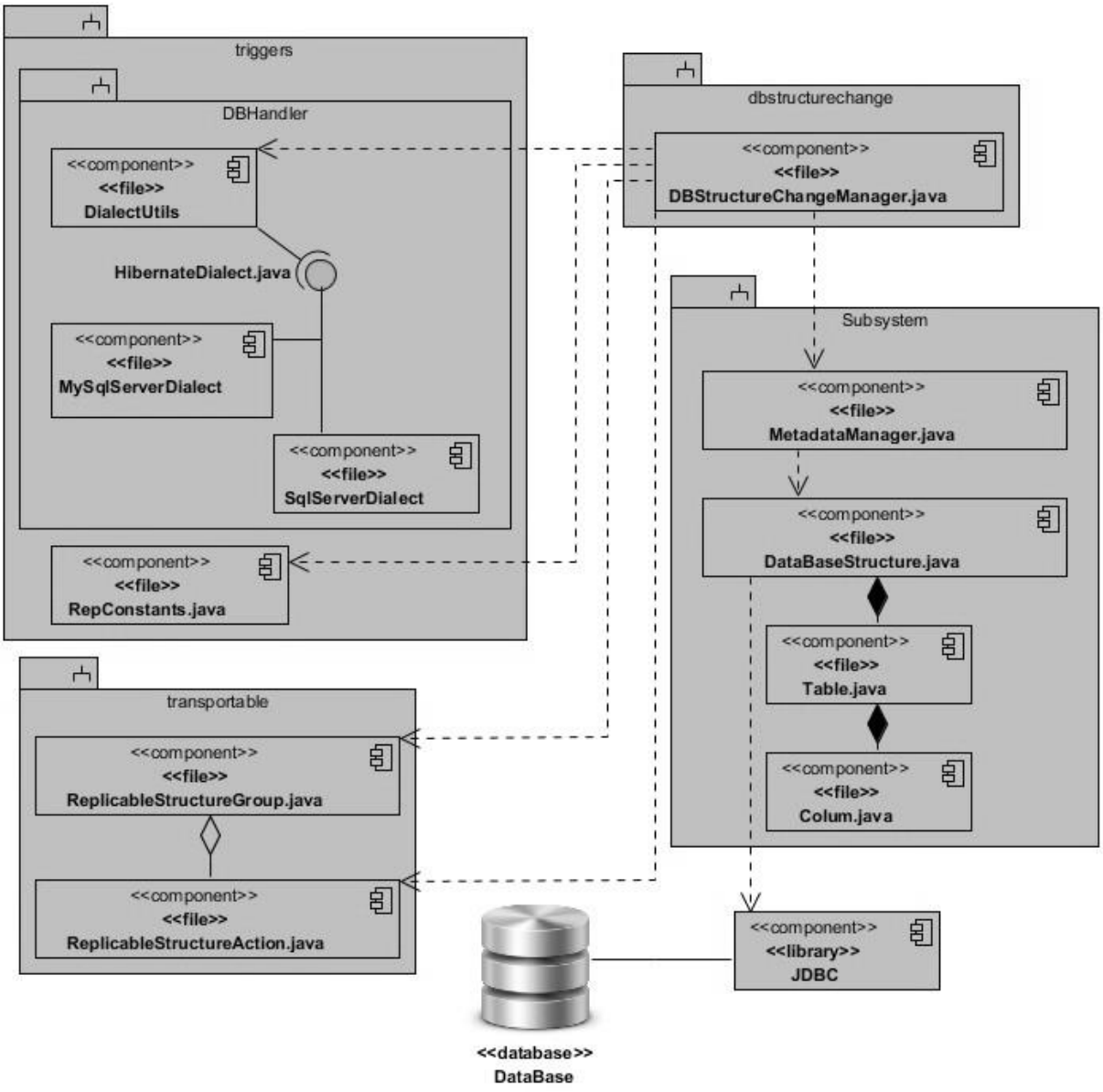


Figura 8 Diagrama de componentes para el sub-sistema Capturador de cambios del Replicador de datos REKO.

Fuente: elaboración propia.

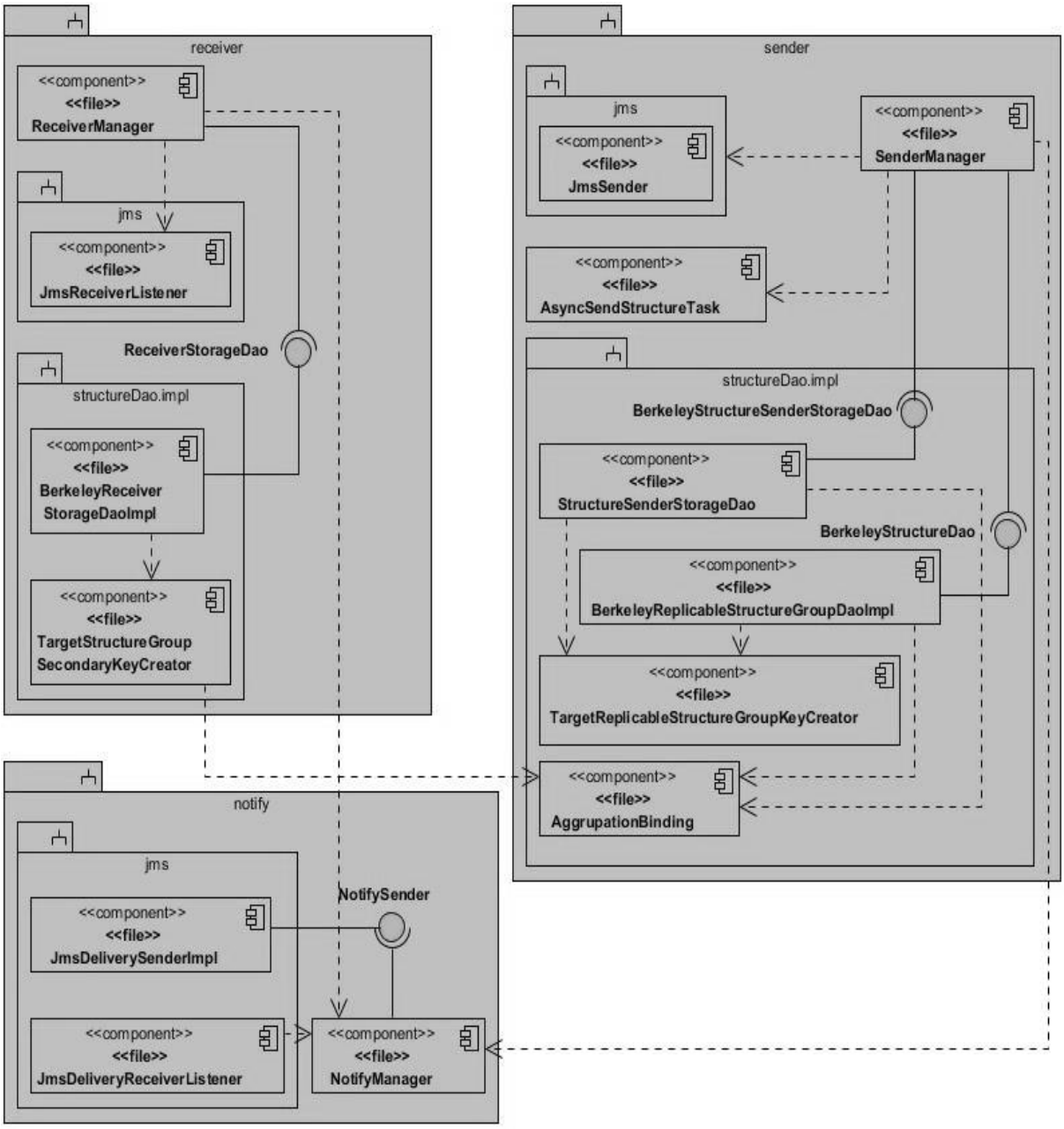


Figura 9 Diagrama de componentes sub-sistema Distribuidor de cambios del Replicador de datos REKO.
Fuente: elaboración propia.

3.2 Código fuente

El código fuente se representa como un conjunto de líneas de textos, que son los pasos que debe seguir la computadora para realizar dicho programa; por lo que es en el código fuente, donde se encuentra escrito el funcionamiento de la computadora. El código fuente de un programa está escrito en un lenguaje de programación determinado, sin embargo, este tipo de lenguaje no puede

ser ejecutado directamente por el computador, sino que debe ser traducido a otro lenguaje que el ordenador pueda ejecutar más fácilmente (5).

Estándar de codificación Java

Los estándares de codificación pueden ser empleados durante el desarrollo de software sobre el lenguaje Java permitiendo el fácil entendimiento, mantenimiento y legibilidad de la aplicación (48).

A continuación, se muestran ejemplos de estándares de codificación Java que se utilizaron en el desarrollo del proceso de réplica de estructuras entre BD gestionadas con Microsoft SQL Server y MySQL para el Replicador de datos REKO. Otros de los estándares de codificación empleados se encuentran en el Anexo 19.

Clases e Interfaces: para estos identificadores se usó la variante *UpperCamelCase*. Todas las palabras que componen a dichos identificadores deben tener la primera letra en mayúscula. Si el nombre es compuesto, cada palabra componente deberá comenzar con mayúscula, ejemplos:

- **public class** DBStructureChangeManager {}

- **public class** SqlServerDialect {}

- **public class** MySqlServerDialect {}

- **public interface** HibernateDialect {}

Métodos: para este identificador se utilizó la variante *lowerCamelCase*. Los métodos deben ser verbos escritos en minúsculas. Cuando el método esté compuesto por varias palabras cada una de ellas tendrá la primera letra en mayúscula.

- **public String** createTable (Table tableName);

- **public String** deleteTupleOfMirrorTable (int id);

- **public String** getCantActionstoReplics ();

3.3 Evaluación del diseño aplicando métricas de software

Las métricas de software son medidas cuantitativas que le permite a los desarrolladores obtener conocimiento acerca de la eficacia del proceso del software y de los proyectos que se efectúan, usando el proceso como marco conceptual. Se reúnen datos básicos de calidad y productividad que luego se analizan y se comparan con promedios anteriores y se valoran para comprobar si

han ocurrido mejoras en calidad y productividad. Las métricas se utilizan también para puntualizar áreas problemáticas, de forma que puedan desarrollarse remedios y mejorar el proceso de software (28).

Se aplicaron las métricas Tamaño Operacional de la Clase y Relaciones entre Clases para evaluar la calidad del diseño del proceso y su relación con los atributos de calidad definidos a continuación:

Responsabilidad: consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.

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 (28).

3.3.1 Métrica Tamaño Operacional de Clase (TOC)

La métrica TOC representa el número de métodos que tiene una clase. En la tabla 7, se muestra la relación que existe entre los atributos de calidad y la forma en que son afectados por esta métrica (49).

Tabla 7 Tamaño operacional de clases

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:(49)

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

Tabla 8 Criterios de evaluación para la métrica TOC

Atributo	**Categoría**	**Criterio**
Responsabilidad	Baja	TOC <= Promedio
	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:(49)

Resultados obtenidos en la aplicación de la métrica TOC

Luego de aplicar la métrica TOC se obtuvieron los resultados de que el diseño propuesto tiene una calidad aceptable, pues el 72 % de las clases poseen una cantidad de procedimientos menor o igual al promedio general de 63,5, lo cual arroja que las evaluaciones sean positivas en los atributos de calidad involucrados (responsabilidad, complejidad de implementación y reutilización). Para una mayor comprensión ver instrumento de evaluación de la métrica TOC en el Anexo 20.

La figura 10 muestra los resultados de la evaluación de la métrica TOC para los atributos anteriormente mencionados.

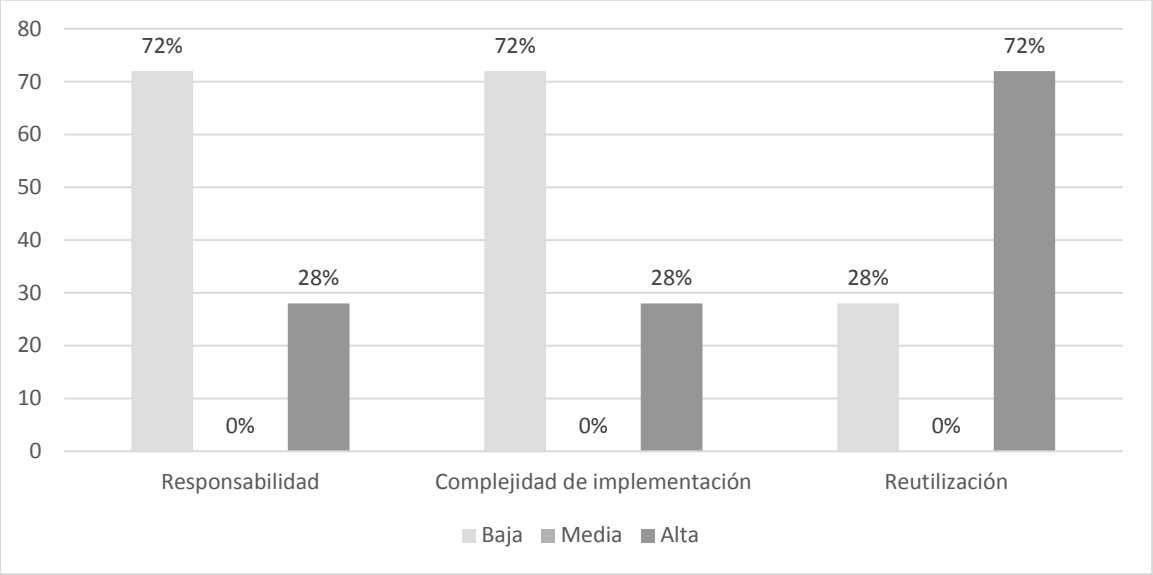


Figura 10 Resultados de la evaluación de la métrica TOC

Fuente: elaboración propia

3.3.2 Métrica Relaciones entre Clases (RC)

La métrica RC representa la cantidad de relaciones de uso de una clase con otra. En la tabla 9, se muestra la relación que existe entre los atributos de calidad y la forma en que son afectados por esta métrica (49).

Tabla 9 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:(49)

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

Tabla 10 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
	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:(49)

Resultados obtenidos en la aplicación de la métrica RC

Luego de aplicar la métrica RC se obtuvieron los resultados de que el diseño propuesto tiene una calidad aceptable, pues el 86 % de las clases poseen una cantidad de relaciones de uso menor o igual al promedio general de 1,28, lo cual arroja que las evaluaciones sean positivas en los atributos de calidad involucrados (acoplamiento, complejidad de mantenimiento, cantidad de pruebas y reutilización). Para una mayor comprensión ver instrumento de evaluación de la métrica RC en el Anexo 21.

La figura 11 muestra los resultados de la evaluación de la métrica RC para los atributos anteriormente mencionados.

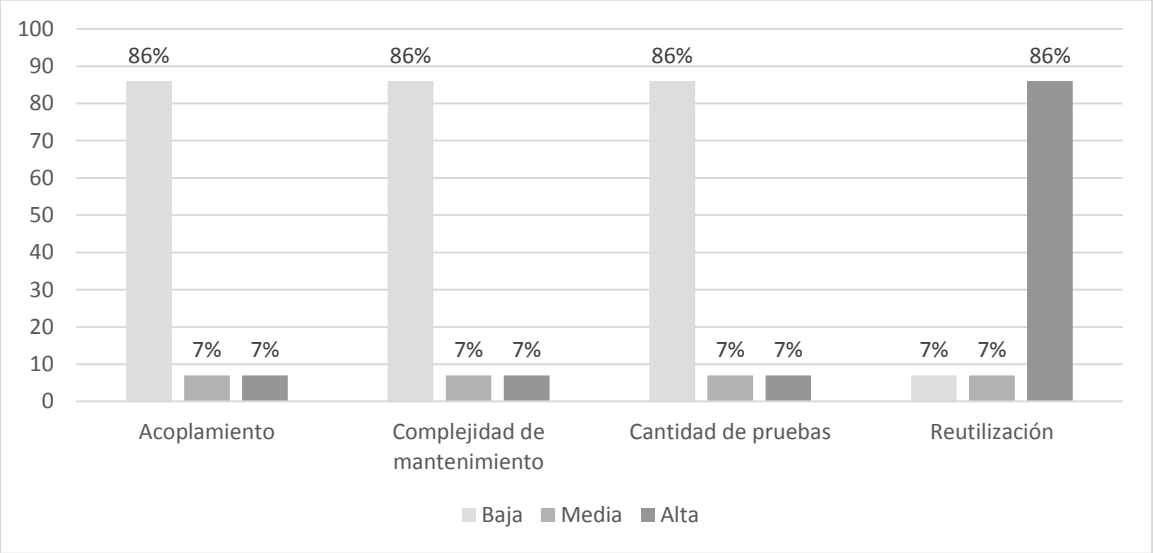


Figura 11 Resultados de la evaluación de la métrica RC

Fuente: elaboración propia

3.3.3 Matriz de inferencia de indicadores de calidad

La matriz de 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 (49).

En la tabla 11 se muestra que los resultados obtenidos para cada atributo de calidad medido son positivos:

Tabla 11 Resultados de la evaluación de la relación atributo/métrica

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

Resultados obtenidos en la aplicación de las métricas de software

Luego de evaluar el proceso realizado utilizando las métricas TOC y RC, se puede comprobar que los resultados obtenidos de los atributos de calidad son positivos ya que todos mantienen un buen comportamiento:

- El grado de responsabilidad del sistema es bajo, lo que significa 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, permitiendo que sus funcionalidades puedan 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 cual 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.

3.4 Pruebas del software

Las pruebas de software consisten en la verificación dinámica del comportamiento de un programa, para asegurarse que arroja el resultado esperado y así garantizar la calidad del software. Además, representa una revisión final de las especificaciones de los requisitos del sistema. Este proceso tiene como objetivos:(45)

- Demostrar al desarrollador y al cliente que el software satisface sus requisitos.
- Descubrir defectos en el software en el que el comportamiento de este es incorrecto, no deseable o no cumple su especificación.

Para lograr el objetivo de las pruebas de software es necesario trazar y ejecutar una serie de pasos (pruebas de unidad, integración, validación y sistema). Las pruebas de unidad e integración realizan la verificación funcional de cada componente y su incorporación 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 (45).

Con el objetivo de validar el funcionamiento de la solución implementada se realizaron una serie de pruebas, con las cuales se pudo confirmar que el sistema mantiene la coherencia de los datos que se replican en el proceso Las pruebas empleadas fueron pruebas de unidad, de sistema y de aceptación.

3.4.1 Pruebas de unidad

La prueba de unidad orienta los esfuerzos de verificación en la unidad más pequeña del diseño de software: el componente o módulo de software. La complejidad de las pruebas y los errores que descubren están limitados por el ámbito restringido que se establece para la prueba de unidad. Las pruebas de unidad se enfocan en la lógica de procesamiento interno y de las estructuras de datos dentro de las fronteras de un componente. Este tipo de pruebas puede realizarse en paralelo para múltiples componentes y está orientada a la técnica de prueba caja blanca (28).

Caja blanca

La prueba de caja blanca, conocida también como prueba de caja de vidrio, es una filosofía de diseño de casos de prueba que usa la estructura de control descrita como parte del diseño a nivel de componentes para derivar casos de prueba. Al usar los métodos de prueba de caja blanca, puede derivar casos de prueba que:(28)

- Garanticen que todas las rutas independientes dentro de un módulo se revisaron al menos una vez.
- Examinen todas las decisiones lógicas en sus lados verdadero y falso.
- Ejecuten todos los bucles en sus límites y dentro de sus límites operacionales.
- Examinen estructuras de datos internas para garantizar su validez.

Se eligió la técnica de camino básico para la realización de la prueba de caja blanca ya que esta técnica proporciona el número mínimo de pruebas que se realiza por adelantado.

Prueba de ruta básica

La prueba de ruta o trayectoria básica es una técnica de prueba de caja blanca que permite al diseñador de casos de prueba obtener una medida de complejidad lógica de un diseño de procedimiento y usar esta medida como guía para definir un conjunto básico de rutas de

ejecución. Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecute por lo menos una vez cada sentencia del programa (28).

Los pasos que se siguen para aplicar esta técnica son:(28)

- Partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
- Se calcula la complejidad ciclomática del grafo.
- Se determina un conjunto básico de caminos independientes.
- Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

A continuación, se muestra el caso de prueba guiado por los pasos anteriores del método **createTriggersSqlServer()**, el cual se encuentra en la clase **DBStructureChangeManager**. El objetivo de este método es construir los *triggers* dentro de la BD de Microsoft SQL Server para capturar los cambios que ocurran en ella. Este método se eligió por la importancia que tiene para el proceso desarrollado.

Tabla 12 Resultados de la prueba de caja blanca

	DBStructureChangeManager. createTriggersSqlServer ()
0	**public void** createTriggersSqlServer(HibernateDialect dialect, Connection con) **throws** DataAccessException, SQLException{
1	**if** (!configurationManager.getAllStructureConfigurations().isEmpty()) {
2	List<DBConfiguration> structureConfigurations = configurationManager .getAllStructureConfigurations();
3	**for** (DBConfiguration structureConfig : structureConfigurations) {
4	String target = ""; String id = structureConfig.getId(); List<Node> listNode = **new** LinkedList<Node>();
5	**if** (id.startsWith("label_")) {
6	listNode = registrationManager.getNodesByLabelId(id);
7	**for** (**int** j = 0; j < listNode.size(); j++) {
8	**if** (j < listNode.size() - 1)
9	target += listNode.get(j).getId() + ",";
10	**Else** target += listNode.get(j).getId(); }

	}
11	**Else** target = id;
12	String createStructureCaptureTriggerCreateDrop = DialectUtils.*createStructureCaptureTriggerCreateDropSqlServer*(dialect, target); String createStructureCaptureTriggerAlter = DialectUtils.*createStructureCaptureTriggerAlterSqlServer*(target, dialect);
13	**try** {
14	MetaDataUtils.*runDDL*(con, createStructureCaptureTriggerCreateDrop); MetaDataUtils.*runDDL*(con, createStructureCaptureTriggerAlter); }
15	**catch** (SQLException e) {

16	logger.error("*** ERROR AL CREAR LAS ESTRUCTURAS DE CONTROL ***");
	con.rollback();
	con.close();
	e.printStackTrace();
	}

17	} }}

Fuente: elaboración propia

Seguidamente se da paso a la construcción del grafo (ver figura 12) correspondiente al código fuente expuesto anteriormente.

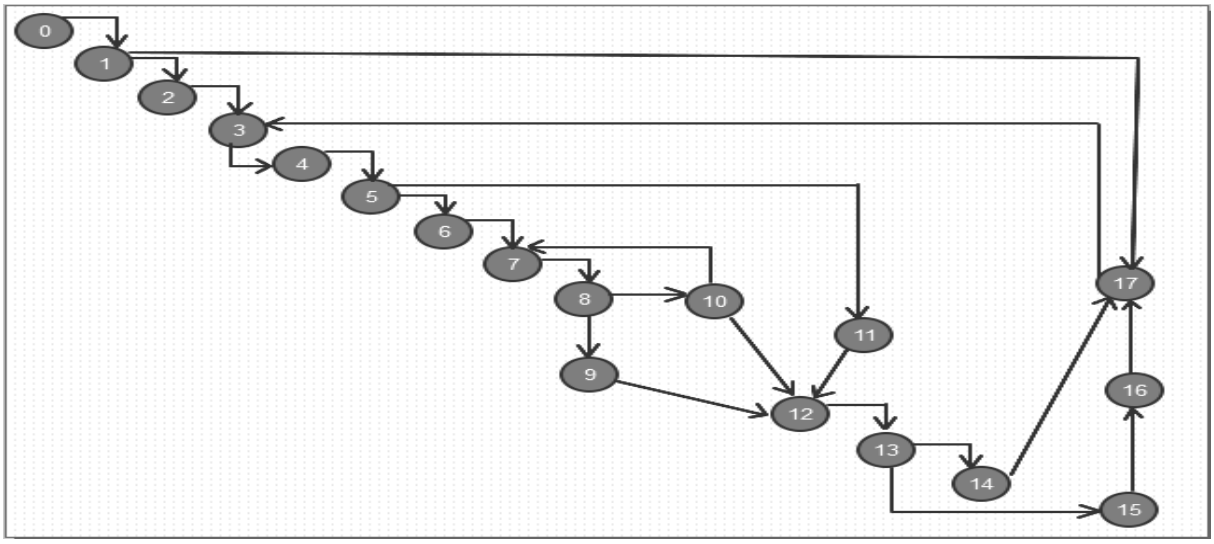


Figura 12 Flujo manual correspondiente al método createTriggersSqlServer ()

Fuente: elaboración propia

Construido el grafo se realiza el cálculo de la complejidad ciclomática mediante las tres fórmulas explicadas a continuación, las cuales deben arrojar el mismo resultado para confirmar que el cálculo de la complejidad es correcto (ver tabla 13).

Tabla 13 Complejidad ciclomática del método createTriggersSqlServer ()

DBStructureChangeManager. buildingReplicableStructureGroup()		
V(G) = R	**V(G) = A - N + 2**	**V(G) = P + 1**
R = 7 (regiones del grafo**)**	**A = 23 (**aristas**)** **N = 18 (**nodos**)**	**P = 6 (**nodos predicados**)**
	V(G) = 23 - 18 + 2	**V(G) = 6 + 1**
V(G) = 7	**V(G) = 7**	**V(G) = 7**

Fuente: elaboración propia

Los cálculos efectuados mediante las tres fórmulas anteriores dan el mismo resultado V(G) = 7, lo cual indica que existen siete posibles caminos por donde el flujo puede circular y establece el número de pruebas que se deben efectuar para asegurar que se ejecute cada sentencia al menos una vez.

Los caminos básicos se presentan a continuación:

- **Ruta 1**: **{**0, 1, 17**}**
- **Ruta 2**: **{**0, 1, 2, 3, 4, 5, 11, 12, 13, 14, 17**}**
- **Ruta 3**: **{**0, 1, 2, 3, 4, 5, 11, 12, 13, 15, 16, 17**}**
- **Ruta 4**: **{**0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14, 17**}**
- **Ruta 5**: **{**0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 17**}**
- **Ruta 6**: **{**0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 15, 16, 17**}**
- **Ruta 7**: **{**0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 15, 16, 17**}**

Los casos de prueba que resguardan los caminos independientes presentados son los siguientes:

Tabla 14 Casos de pruebas de los caminos independientes

No	**Entrada**	**Resultado**
1	No hay configuraciones de réplicas guardadas.	No se crean los *triggers* en la BD.

2	No obtiene el SQL para crear los *triggers* en la BD.	Error al crear las estructuras de control
3	No existen destinos registrados.	Se asigna el id de configuración al destino.
4	Se recorre la lista de destinos y se les pasa por parámetros a los *triggers*.	Se crean los *triggers* en la BD.
5	Solo existe un destino y se le pasa por parámetro a los *triggers*.	Se crean los *triggers* en la BD.
6	Se recorre la lista de destinos pero no se ejecuta el SQL para crear los *triggers*.	Retorna una excepción.
7	Solo existe un destino pero no se ejecuta el SQL para crear los *triggers*.	Retorna una excepción.

Fuente: elaboración propia

Resultados de las pruebas de caja blanca

Se aplicaron pruebas de caja blanca al código fuente del proceso, que demostraron el buen funcionamiento de los **RF** (**RF1**, **RF2**, **RF3**, **RF4** y **RF5**), alcanzándose las respuestas esperadas.

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 crean los *triggers.*
- Se registran los cambios de estructura capturados en la tabla de control, pero no se especifica el destino de esos cambios.
- Error con las conexiones.

En la segunda iteración fueron resueltas tres NC y se identificó dos nuevas NC:

- No se crea el *trigger* para los cambios de tipo ALTER.
- Se registran los cambios de estructura capturados en la tabla de control, pero no se guardan todos los detalles referentes a los cambios de tipo ALTER.

En una tercera iteración se resolvió las NC detectadas en la segunda iteración y no se detectaron NC nuevas.

Las pruebas de caja blanca realizadas al sistema detectaron un total de 5 no conformidades. En la Figura 13 se muestra un gráfico que refleja la cantidad de no conformidades detectadas por iteración.

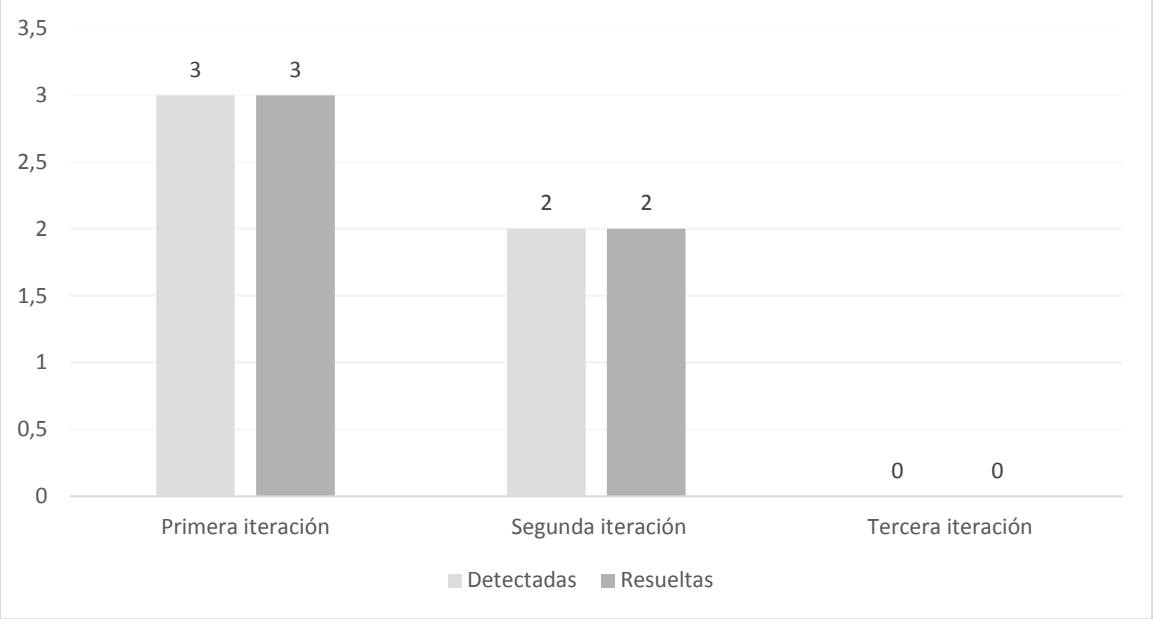


Figura 13 Cantidad de no conformidades de caja blanca

Fuente: elaboración propia

3.4.2 Pruebas del sistema

Las pruebas del sistema tienen como propósito principal ejercitar por completo el sistema basado en computadora, para verificar que los elementos del sistema se hayan integrado de manera adecuada y que se realicen las funciones asignadas. Estas validan el software una vez que se incorporó en un sistema más grande. Cada paso de la prueba se logra a través de una serie de técnicas de prueba sistemáticas que ayudan en el diseño de casos de prueba (28).

Caja negra

Las pruebas de caja negra también llamadas pruebas de comportamiento representan las pruebas que se llevan a cabo en la interfaz del software. Se enfocan en los requerimientos funcionales del software; es decir, permiten derivar conjuntos de condiciones de entrada que revisarán por completo todos los requerimientos funcionales para un programa. Las pruebas de caja negra intentan encontrar errores en las categorías siguientes:

- Funciones incorrectas o faltantes.

- Errores de interfaz.
- Errores en las estructuras de datos o en el acceso a bases de datos externas.
- Errores de comportamiento o rendimiento.
- Errores de inicialización y terminación (28).

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

Método de partición equivalente

La partición de equivalencia es un método de prueba de caja negra que divide el dominio de entrada de un programa en clases de datos de los que pueden derivarse casos de prueba. Permite examinar los valores válidos y no válidos de las entradas existentes en el software. Además, se esfuerza por definir un caso de prueba que revele ciertas clases de errores y reduce el número total de casos de prueba que deben desarrollarse (28).

Se expone a continuación los resultados obtenidos al aplicar la prueba de caja negra mediante el método de partición equivalente (ver Tabla 14), este método se aplicó al RF1 demostrando resultados satisfactorios.

Tabla 15 Resultados de la prueba uno de caja negra sobre la HU 1

Caso de Prueba	
Código: SC_1	**Historia de usuario**: 1
Nombre: Crear objetos de la BD para capturar cambios DDL	
Descripción: prueba para la funcionalidad de crear objetos de la BD para capturar cambios DDL.	
Condiciones de Ejecución: • El usuario debe estar autenticado. • Establecer la configuración de BD para Microsoft SQL Server o MySQL 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 estructuras de control en la BD en caso de no existir, inicia la captura de los cambios de esquema si existe una configuración de réplica de estructura y muestra la configuración establecida.	

Resultado Esperado: el sistema guarda la configuración para programar la captura y envío de los cambios de estructuras, cuando no se está replicando cambios de esquemas.
Evaluación de la Prueba: bien.

Fuente: elaboración propia

Resultados de las pruebas de caja negra

Para las pruebas de caja negra se realizaron tres iteraciones. En la primera iteración se detectaron dos no conformidades (NC):

- No se crearon las estructuras de control en la BD al guardar la configuración establecida.
- No se crearon correctamente los objetos de captura en la BD al guardar la configuración.
- Error con las conexiones.
- Los objetos de captura no capturan correctamente los cambios realizados.

En una segunda iteración se resolvieron tres NC detectadas en la primera iteración y se detectaron más NC.

- Las estructuras de control en la BD ya existen, pero siguen creándose.
- No se crearon correctamente los objetos de captura en la BD al guardar la configuración.
- Los objetos de captura creados en la BD capturan cambios sobre las estructuras de control.
- Los objetos de captura no capturan correctamente los cambios realizados de tipo ALTER.

En una tercera iteración se resolvió las NC detectadas en la segunda iteración y no se detectaron NC nuevas.

Las pruebas de caja negra realizadas al sistema detectaron un total de 7 no conformidades. En la Figura 14 se muestra un gráfico que refleja la cantidad de no conformidades detectadas por iteración.

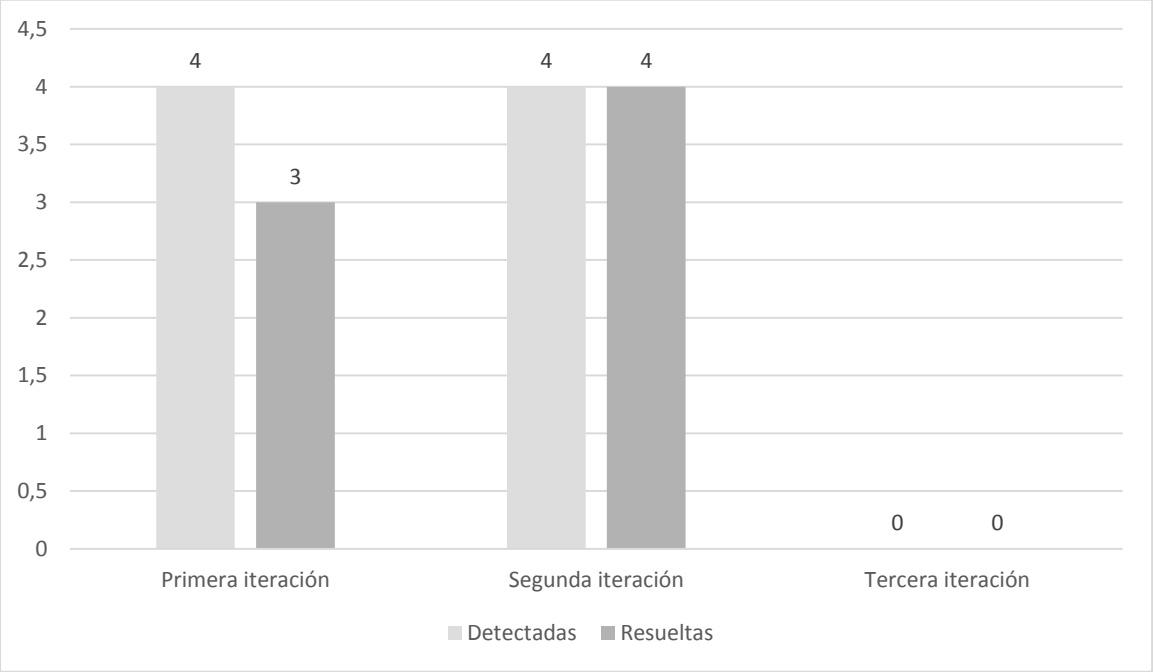


Figura 13 Cantidad de no conformidades de caja negra

Fuente: elaboración propia

3.4.3 Pruebas de aceptación

Las pruebas de aceptación también llamadas pruebas del cliente, son especificadas por el cliente y se enfocan en las características y funcionalidades generales del sistema que son visibles y revisables por parte del cliente. Las pruebas de aceptación se realizan a partir de las historias de los usuarios que se han efectuado como parte de la liberación del software (28).

3.5 Resultados obtenidos

Al proceso de réplica de estructura entre BD gestionadas con Microsoft SQL Server y MySQL le fue aplicado pruebas de software para detectar las NC presentes en la propuesta de solución.

Se le realizaron 3 iteraciones a través del uso del método de caja blanca, el cual arrojó los siguientes resultados:

- En la primera iteración se encontraron 3 NC de tipo funcional, en la segunda iteración, se resolvieron las NC detectadas en la primera iteración y se encontraron 2 NC nuevas que fueron resueltas en una tercera iteración donde no se detectaron NC nuevas, para un total de 5 NC después de haber concluido la ejecución de las mismas.

También se realizaron 3 iteraciones utilizando el método de caja negra mediante la técnica de partición equivalente, las cuales arrojaron los siguientes resultados:

- En la primera iteración se encontraron 4 NC de las cuales 3 fueron resueltas en una segunda iteración y encontrado 3 NC nuevas, que fueron resueltas en una tercera iteración donde no se detectaron NC nuevas, para un total de 7 NC después de haber concluido la ejecución de las mismas.

3.6 Consideraciones del capitulo

- En el presente capítulo se caracterizaron los estándares de codificación para la implementación de las clases y los métodos definidos en la fase de análisis y diseño.
- Se aplicaron las métricas de software TOC y RC que validaron la calidad del modelo de diseño propuesto como aceptable.
- Se realizaron las pruebas de software y se corrigieron los errores de ejecución de los procesos no satisfactorios.

CONCLUSIONES GENERALES

Luego de realizar el análisis, el diseño, la implementación y las pruebas al proceso que permite la réplica de estructura entre BD gestionadas con Microsoft SQL Server y MySQL, se arribaron a las siguientes conclusiones:

- El empleo de la metodología, herramientas y tecnologías definidas por el proyecto al cual se realiza el proceso permitió obtener un producto acorde los estándares de desarrollo del Replicador de Datos REKO.
- Quedó implementado en REKO un nuevo mecanismo de captura de cambios de estructura para BD gestionadas con Microsoft SQL Server y MySQL que mantiene la coherencia de los datos que se replican en el proceso.
- La aplicación de métricas de software y la realización de pruebas a la solución permitieron detectar y corregir a tiempo los errores existentes, obteniéndose un producto funcional con la calidad requerida.

RECOMENDACIONES

Adicionar al sistema la replicación de otras estructuras como vistas, secuencias, triggers entre otras cuando se extienda el metadato del Replicador de datos REKO.

REFERENCIAS BIBLIOGRÁFICAS

1. FRASSIA, M. *INTRODUCCIÓN A LAS BASES DE DATOS.* Disponible en: http://www.cursogis.com.ar/BasesP/Zip/Base_Clase1.pdf.

2. GUTIÉRREZ DÍAZ, A. *BASES DE DATOS DISTRIBUIDAS.* Disponible en: http://cursos.aiu.edu/Base%20de%20Datos%20Distribuidas/pdf/Tema%201.pdf.

3. COBO, Á. *Diseño y programación de bases de datos.* Editorial Visión Libros, 2007. ISBN 8499831478.

4. JEREZ, G. R. L. Componente para el envío y recepción de datos del proceso de réplica de estructura para el software de replicación Reko. 2013, nº

5. MARQUÉS, M. *Bases de Datos.* publicado el: Enero 2009 de 2009, última actualización: Enero 2009. Disponible en: http://www3.uji.es/~mmarques/apuntes_bbdd/apuntes.pdf.

6. GÓMEZ FUENTES, M. D. C. *NOTAS DEL CURSO BASES DE DATOS.* 2013. ISBN 978 - 607 - 477 - 8 80 - 9.

7. CAMPS PARÉ, R.; CASILLAS SANTILLÁN, L. A.*, et al. Bases de datos.* 2005. ISBN 84-9788-269-5.

8. GÓMEZ BALLESTER, E.; MARTÍNEZ BARCO, P.*, et al. Bases de Datos 1.* Disponible en: https://rua.ua.es/dspace/bitstream/10045/2990/1/ApuntesBD1.pdf

9. JAIME, A. *Bases de Datos Distribuidas (BDD).* publicado el: 2005 de 2005, última actualización: 2005. Disponible en: http://www.unirioja.es/cu/arjaime/Temas/09.Distribuidas.pdf.

10. MENDOZA, M. E. *BASES DE DATOS DISTRIBUIDAS.* Disponible en: http://univirtual.unicauca.edu.co/moodle/pluginfile.php/18662/mod_resource/content/0/Materiales/clase_10/05_-_2004-04-21-BD_Distribuidas.pdf.

11. GARCÍA MEDINA, M. *BASES DE DATOS.* publicado el: 2013 de 2013, última actualización: 2013. Disponible en: http://e-learning.cecar.edu.co/RecursosExternos/e-book/BasesdeDatos.pdf.

12. GUTIÉRREZ CARREÓN, G. A. *Lenguaje de Consultas SQL*. En

13. CARÁMBULA, L. *DDL - Lenguaje de Definición de Datos*. En 30/8/2010.2010.

14. *SQL Server Replication | Microsoft Docs*. Disponible en: https://docs.microsoft.com/es-es/sql/relational-databases/replication/sql-server-replication.

15. URBANO, R. *Oracle Streams Replication Administrator's.* 2008.

16. RODRÍGUEZ-TASTETS, A. *Introducción a Bases de Datos*. En 3/8/2007.2007.

17. ROJAS, D. *Propuesta metodológica para el desarrollo y la elaboración de estadísticas ambientales en países de América Latina y el Caribe.* 2005.

18. ECHEGOYEN, G. *Registros administrativos, calidad de los datos y credibilidad pública: presentación y debate de los temas sustantivos de la segunda reunión de la Conferencia Estadística de las Américas de la CEPAL.* 2003.

19. SATURNO HERNÁNDEZ, P. J. *Qué, Cómo y Cuándo Monitorizar: Marco Conceptual y Guía Metodológica.* publicado el: 19/10/2010 de 2010, última actualización: 19/10/2010. Disponible en: http://upchmed.pe/red_cochrane_peru/wp-content/uploads/2012/09/Taller_6_Seguridad_del_Paciente_Lectura_Sugerida_13_Qu%C3%A9_C%C3%B3mo_y_Cu%C3%A1ndo_Monitorizar_Dr.Garc%C3%ADaElorrio1.pdf.

20. MOLERO, X.; JUIZ, C.*, et al. Evaluación Y Modelado Del Rendimiento De Los Sistemas Informáticos.* 2004. ISBN 84-205-4093-5.

21. *SymmetricDS Multi-Master Data Sync*. Disponible en: http://www.symmetricds.org/about/overview.

22. AZCUY, R. A. Módulo para la configuración y monitorización de réplicas con la herramienta SymmetricDS para la arquitectura Xalix. 2017, nº

23. MANSO GUERRA, Y.; JOSÉ ALTUNA, E.*, et al.* Experiencias en el uso del symmetric para la replicación de datos en la plataforma educativa ZERA. 29/09/2015 2015, nº Disponible en: http://scielo.sld.cu/pdf/rcci/v9n4/rcci14415.pdf.

24. *Data Integration - BackOffice Associates*. Disponible en: http://www.hitsw.com/localized/spanish/products_services/dbmoto/dbmoto_dsheet.html.

25. BROWNE, C. *Slony-I 2.1.4 Documentation.* publicado el: 16/8/2013 de 2013, última actualización: 16/8/2013. Disponible en: http://slony.info/adminguide/2.1/doc/adminguide/slony.pdf.

26. *ORACLE.* 2017, Disponible en: https://docs.oracle.com/cd/B28359_01/server.111/b28321/strms_over.htm#i1006084.

27. BARRIENTOS TRENAL, N. W. *Proceso de réplica de estructura entre bases de datos gestionadas en Oracle para el Replicador de datos Reko.* 2017 de 2017

28. PRESSMAN, R. S. y TROYA, J. M. Ingeniería del software. 1988, nº

29. EDEKI, C. Agile Unified Process. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE*, 2013, vol. 1, nº 3,

30. SÁNCHEZ, T. R. Metodología de desarrollo para la Actividad productiva de la UCI. 2014, nº

31. RUMBAUGH, J.; JACOBSON, I.*, et al. El Lenguaje Unificado de Modelado Manual de Referencia.* 2002.

32. BAQUERO HERNÁNDEZ, L.; MENDOZA PEÑA, D.*, et al.* Extensión de la herramienta Visual Paradigm for UML para la evaluación y corrección de Diagramas de Casos de Uso. *Serie Científica de la Universidad de las Ciencias Informáticas*, April 30, 2016 2016, vol. 9, nº p. 1-14. Disponible en: https://www.researchgate.net/publication/301750902_Extension_de_la_herramienta_Visual_Paradigm_for_UML_para_la_evaluacion_y_correccion_de_Diagramas_de_Casos_de_Uso.

33. WEITZENFELD, A. *Ingeniería de software orientada a objetos con UML, Java e Internet.* 2009.

34. GUTIÉRREZ, J. *Eclipse y Java.* publicado el: 14/7/2004 de 2004, última actualización: 14/7/2004. Disponible en: https://www.uv.es/~jgutierr/MySQL_Java/TutorialEclipse.pdf.

35. *Apache ActiveMQ ™ -- Index*. Disponible en: http://activemq.apache.org/.

36. *Apache Tomcat® - Welcome!* Disponible en: http://tomcat.apache.org/.

37. *Biblioteca de Microsoft SQL Server*. Disponible en: https://msdn.microsoft.com/es-es/library/bb545450.aspx.

38. *MySQL :: MySQL 5.7: 3x Faster*. Disponible en: https://www.mysql.com/products/enterprise/database/.

39. LARMAN, C. y VALLE, B. M. *UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado.* Pearson Educación, 2003. ISBN 9788420534381.

40. ARIAS CHAVES, M. La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software. *InterSedes: Revista de las Sedes Regionales*, 2005, vol. 6, nº 10, ISSN 2215-2458.

41. ORJUELA DUARTE, A. y ROJAS, M. Las metodologías de desarrollo ágil como una oportunidad para la ingeniería del software educativo. *Revista Avances en Sistemas e Informática*, 2008, vol. 5, nº 2, ISSN 1657-7663.

42. ESPAÑA, S. y FERNANDO, H. *Documentación y análisis de los principales frameworks de arquitectura de software en aplicaciones empresariales.* Facultad de Informática, 2016.

43. REYNOSO, C. y KICILLOF, N. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004,

44. ORACLE. *Core J2EE Patterns - Data Access Object* Disponible en: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html.

45. SOMMERVILLE, I. *Ingeniería del software.* Pearson Educación, 2005. ISBN 8478290745.

46. TORRES, P. L. Desarrollo de Software Orientado a Objeto usando UML. *Universidad Politecnica de Valencia (UPV)–España*, 2004, nº

47. GIRALDO, G. L.; ACEVEDO, J. F.*, et al.* Una ontología para la representación de conceptos de diseño de software. *Revista Avances en Sistemas e Informática*, 2011, vol. 8, nº 3, ISSN 1657-7663.

48. FLOWER. *Java Foundations: Java - Estándares de programación.* vol. 2018-05-05 23:07:43, Disponible en: http://javafoundations.blogspot.com/2010/07/java-estandares-de-programacion.html.

49. BARYOLO, O. G. Solución Informática de Autorización en Entornos Multientidad y Multisistema. *Universidad de las Ciencias Informáticas*, 2010, nº

ANEXOS

Anexo 1: descripción del **RnF1** Fiabilidad, sub-atributos Tolerancia a fallos y Recuperabilidad.

Atributo de Calidad	Fiabilidad.
Sub-atributos/Sub-característica	Tolerancia a fallos y recuperabilidad.
Objetivo	Capacidad del producto para operar según lo previsto en presencia de fallos de hardware o software. Capacidad del producto para recuperar los datos directamente afectados y restablecer el estado deseado del sistema en caso de interrupción o fallos.
Origen	Proveedor de requisitos.
Artefacto	El sistema y el código fuente.
Entorno	El sistema desplegado y existen fallos en la red y eléctricos.
Estímulo **<<1>>. <<a>> <Recuperación ante un fallo de conexión al servidor de mensajería >**	**Respuesta: Flujo de eventos (Escenarios)**
Fallos en la red	1. El sistema se encuentra capturando y enviando acciones de réplica hacia el destino. 2. El sistema intenta enviar los datos, pero detecta que está desconectado y persiste en la base de datos local todas las acciones a replicar hacia el nodo destino. 3. El sistema se conecta al servidor de mensajería y envía las acciones cuando se reestablece la comunicación con el servidor de mensajería. 4. El sistema concluye el proceso antes afectado satisfactoriamente

<<1>>. <<b>> <Recuperación ante un fallo del fluido eléctrico >	**Respuesta: Flujo de eventos (Escenarios)**
Fallos eléctricos.	1. El sistema se encuentra capturando y enviando acciones de réplica hacia el destino. 2. El sistema falla por ausencia del fluido eléctrico. 3. El sistema se reestablece antes el fallo eléctrico iniciando de forma automática con el sistema operativo y cargando las últimas acciones no ejecutadas de su base de datos local.
Medida de respuesta	
Navegar en el sistema en presencia de fallos eléctricos y de recursos de red.	

Fuente: elaboración propia

Anexo 2: descripción del **RnF2** Mantenibilidad, sub-atributo Analizabilidad.

Atributo de Calidad	Mantenibilidad.
Sub-atributos/Sub-característica	Analizabilidad.
Objetivo	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.
Origen	Arquitecto de software.
Artefacto	El código fuente.
Entorno	El ambiente de desarrollo del sistema.
Estímulo	**Respuesta: Flujo de eventos (Escenarios)**
<<1>>. <<a>>< Impacto de un determinado cambio sobre el resto del sistema>	

Se evalúa el impacto de un cambio en el sistema.	1. El arquitecto de software identifica los paquetes y clases implicados en el cambio, así como las funciones que se reutilizarán o se crearán para introducir el cambio. 2. Se evalúa el impacto partiendo de los resultados que arrojó el análisis anterior con respecto a la arquitectura del sistema.
<<1>>. <<b>>< Diagnosticar las deficiencias o causas de fallos en el sistema>	**Respuesta: Flujo de eventos (Escenarios)**
Se diagnostica las deficiencias o causas de fallos en el sistema.	1. El arquitecto de software identifica las posibles deficiencias o causas de fallos que se pueden originar en el sistema. 2. Se diagnostican las deficiencias o causas de fallos partiendo de los resultados que arrojó el análisis anterior.
<<1>>. <<c>>< Identificar las partes a modificar>	**Respuesta: Flujo de eventos (Escenarios)**
Se identifica las partes a modificar en el sistema.	1. El arquitecto de software identifica los paquetes y clases implicados en el cambio, así como las funciones que se reutilizarán o se crearán para introducir el cambio.
Medida de respuesta	
Analizar el cambio en el sistema.	

Fuente: elaboración propia

Anexo 3: descripción del **RnF2** Mantenibilidad, sub-atributo Modificabilidad.

Atributo de Calidad	Mantenibilidad.
Sub-atributos/Sub-característica	Modificabilidad.
Objetivo	Capacidad del producto que permite que sea modificado de forma efectiva y eficiente

	sin introducir defectos o degradar el desempeño.
Origen	Arquitecto de software.
Artefacto	El código fuente.
Entorno	El ambiente de desarrollo del sistema.
Estímulo **<<1>>. <<a>>< Capacidad de modificación del sistema >**	**Respuesta: Flujo de eventos (Escenarios)**
La arquitectura del sistema está diseñada para brindar facilidades a la hora de introducir modificaciones en el sistema. Esto permite que se puedan introducir cambios o modificaciones, que no afecten el correcto desempeño del resto de la solución.	NA
Medida de respuesta	
Introducir una modificación al sistema.	

Fuente: elaboración propia

Anexo 4: descripción del **RnF2** Mantenibilidad, sub-atributo Capacidad para ser probado.

Atributo de Calidad	Mantenibilidad.
Sub-atributos/Sub-característica	Capacidad para ser probado.
Objetivo	Facilidad con la que se pueden 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.
Origen	Arquitecto de software.
Artefacto	Diseños de casos de pruebas.
Entorno	El sistema desplegado.
Estímulo	

<<1>>. <<a>>< Facilidad con la que se pueden establecer criterios de prueba para el sistema >	**Respuesta: Flujo de eventos (Escenarios)**
Desde la concepción inicial del sistema se definen y delimitan las funciones según los requisitos funcionales y no funcionales a implementar especificando los argumentos o variables de entrada y salida del sistema, elementos que favorecen el proceso de identificación y elaboración de los distintos escenarios de prueba.	NA
Medida de respuesta	
Escenario de prueba del sistema.	

Fuente: elaboración propia

Anexo 5: descripción del **RnF3** Adecuación funcional, sub-atributo Integridad funcional.

Atributo de Calidad	Adecuación funcional.
Sub-atributos/Sub-característica	Integridad funcional.
Objetivo	Grado en el que el conjunto de funciones cubre todas las tareas y objetivos del usuario especificado.
Origen	Proveedor de requisitos.
Artefacto	El sistema.
Entorno	El sistema desplegado.
Estímulo	**Respuesta: Flujo de eventos (Escenarios)**
<<1>>. <<a>>< Grado en el que el sistema cubre todas las tareas y objetivos especificados >	
El desarrollo del sistema está guiado por las necesidades expresadas por parte de los proveedores de requisitos, dándole total cumplimiento a sus especificaciones declaradas e implícitas.	NA

Medida de respuesta
Analizar las funcionalidades del sistema.

Fuente: elaboración propia

Anexo 6: descripción del **RnF3** Adecuación funcional, sub-atributo Corrección funcional.

Atributo de Calidad	Adecuación funcional.
Sub-atributos/Sub-característica	Corrección funcional.
Objetivo	Grado en que un producto o sistema proporciona los resultados correctos con el grado necesario de precisión.
Origen	Proveedor de requisitos.
Artefacto	El sistema.
Entorno	El sistema desplegado.
Estímulo	**Respuesta: Flujo de eventos (Escenarios)**
<<1>>. <<a>>< Grado en el que el sistema proporciona los resultados correctos con precisión >	
El sistema realiza el proceso de réplica con la exactitud necesaria en cada uno de los nodos de réplica, en caso de que existan errores la solución es capaz de gestionar los conflictos para mantener la integridad de las base de datos implicadas en el proceso de réplica.	NA
Medida de respuesta	
Navegar en el sistema.	

Fuente: elaboración propia

Anexo 7: descripción del **RnF4** Portabilidad, sub-atributo Adaptabilidad.

Atributo de Calidad	Portabilidad.
Sub-atributos/Sub-característica	Adaptabilidad.
Objetivo	Capacidad del producto que le permite ser adaptado de forma efectiva y eficiente a

	diferentes entornos determinados de hardware, software, operacionales o de uso.
Origen	Arquitecto de software.
Artefacto	El sistema.
Entorno	El sistema desplegado.
Estímulo **<<1>>. <<a>>< Capacidad del sistema de adaptarse de forma efectiva a diferentes entornos >**	**Respuesta: Flujo de eventos (Escenarios)**
El sistema está diseñado con tecnologías que permiten que este se adapte a cualquier sistema operativo. El script de inicio de la aplicación permite personalizar los recursos de RAM de uso de la aplicación. El fichero de configuración de las propiedades del sistema permite configurar los parámetros del sistema según las prestaciones que posea la estación de trabajo donde se encuentre instalado.	NA
Medida de respuesta	
El ambiente de despliegue del sistema.	

Fuente: elaboración propia

Anexo 8: descripción del **RnF5** Seguridad, sub-atributo No repudio.

Atributo de Calidad	Seguridad.
Sub-atributos/Sub-característica	No repudio.
Objetivo	Grado en que las acciones o eventos pueden ser probados a haber tenido lugar, por lo que los eventos o acciones no pueden ser repudiados más tarde.
Origen	Arquitecto de software.
Artefacto	El sistema.

Entorno	El sistema desplegado.
Estímulo **<<1>>. <<a>>< Sistema de eventos o trazas >**	**Respuesta: Flujo de eventos (Escenarios)**
El sistema una vez iniciado crea un conjunto de trazas a nivel de ficheros en la raíz de la solución en una carpeta nombrada log donde se almacenan tres archivos: Audit.log: contiene todas las trazas relacionadas con las respuestas de la aplicación al cliente. Error.log: contiene los errores ocurridos en el sistema. System.log: contiene todas las trazas relacionadas con las acciones ejecutadas por el sistema. Este sistema de trazas garantiza el no repudio sobre las acciones realizadas en el sistema.	NA
Medida de respuesta	
Navegar en el sistema.	

Fuente: elaboración propia

Anexo 9: Descripción de la HU2: Capturar los cambios de estructura.

	Historias de usuario
Número: HU 2	**Nombre del requisito**: Capturar los cambios de estructura.
Programador: Jennifer Cáceres Vega y Jose Manuel Blnco Peña	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 30 días
Riesgo en Desarrollo: Plan de riesgos	**Tiempo Real**: 120 horas
Descripción: Para capturar los cambios de estructura del gestor Microsoft SQL Server se crean dos *triggers* en la clase **SqlServerDialect**: el trigger "**trigger_capture_create_drop**" el cual captura los cambios de estructura por la acción *DDL* (*créate y drop*), y el *trigger*	

"**trigger_capture_alter**" que captura los cambios *DDL* (alter); los *triggers* se crean a partir de una configuración de réplica. En el caso de MySQL se traduce un log el cual guardará la hora y la fecha de cada cambio DDL realizado en la BD.
Observaciones: El objeto de captura se creará si existe alguna configuración de réplica de estructura establecida y si la opción "*Replicar cambios de esquemas*" está activada.
Prototipo de interfaz: NA

Fuente: elaboración propia

Anexo 10: Descripción de la HU3: Mostrar en la tabla de control de la BD los cambios capturados.

Historias de usuario	
Número: HU 3	**Nombre del requisito**: Guardar en la tabla de control de la BD los cambios capturados.
Programador: Jennifer Cáceres Vega y Jose Manuel Blnco Peña	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 30 días
Riesgo en Desarrollo: Plan de riesgos	**Tiempo Real**: 120 horas
Descripción: Se implementara el método **createStructureControlTableSQL()** el cual crea la tabla de control en la que ocurrido un cambio en la BD los objetos de captura guardan el cambio y detalles sobre este**.**	
Observaciones:	
Prototipo de interfaz: NA	

Fuente: elaboración propia

Anexo 11: Adicionar dialecto SQL del lenguaje DDL para aplicar cambios.

Historias de usuario	
Número: HU 2	**Nombre del requisito**: Adicionar SQL del lenguaje DDL para aplicar cambios.
Programador: Jennifer Cáceres Vega y Jose Manuel Blnco Peña	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 30 días
Riesgo en Desarrollo: Plan de riesgos	**Tiempo Real**: 120 horas

Descripción: En las clases **SqlServerDialect** y **MySqlServerDialect** se implementarán métodos para reconocer la sentencia SQL correspondiente a cada gestor para así reconocer el cambio y poder aplicarlo.
Observaciones:
Prototipo de interfaz: NA

Fuente: elaboración propia

Anexo 12: Descripción de la clase DBStructureChangeManager.

Descripción de la clase DBStructureChangeManager	
Nombre: DBStructureChangeManager	
Tipo de clase: Controladora	
Atributos	**Tipos**
sqlMetadataManager	**Private**
replicableStructureGroupStorageDao	**Private**
transactionProcessor	**Private**
count	**Private**
Responsabilidades	
Nombre:	buildingReplicableStructureGroup()
Descripción:	Construye el grupo de estructuras replicables que contiene las acciones referentes a los cambios de estructuras que se van a replicar.
Nombre:	getReplicaleStructureActionByResultSet (result: java.sql.ResultSet, dialect:cu.uci.dbreplication.triggers.DBHandler.HibernateDialect): cu.uci.dbreplication.transportable.ReplicableStructureAction
Descripción:	Construye la acción de estructuras replicables con la información almacenada en la tabla espejo a través del result especificado por parámetro, y con la tabla almacenada en el metadata.
Nombre:	getNewReplicableStructureGrupId(): String
Descripción:	Responsable de construir un identificador único para el RSG, incrementando el contador que está como atributo.

Fuente: elaboración propia

Anexo 13: Descripción de la clase SenderManager.

Descripción de la clase SenderManager	
Nombre: SenderManager	
Tipo de clase: Controladora	
Atributos	**Tipos**
logger	**Protected**
sendingStructure	**Private**
splitStructureGroup	**Private**
sender	**Private**
listReplicableStructureGroup	**Private**
asyncSendStructureTask	**Private**
monitoringManager	**Private**
replicableStructureGroupStorageDao	**Private**
structureSenderStorageDao	**Private**
nodesWithReplicationStopped	**Private**
exportConfig	**Private**
Responsabilidades	
Nombre:	afterPropertiesSet()
Descripción:	Método a implementar de la interfaz "**InitializingBean**". Inicia las tareas de ejecución periódica. Garantiza que en caso de interrupciones, las agrupaciones pendientes sean enviadas antes de las nuevas.
Nombre:	addReplicableStructureGroup(aggrupation:cu.uci.dbreplication.transportable.ReplicableStructureGroup)
Descripción:	Adiciona la agrupación especificada por parámetros a la lista de agrupaciones.
Nombre:	sendReplicableStructureGroup ()
Descripción:	Mientras existen agrupaciones las envía.
Nombre:	sendIfReplicableStructureGroupIsAplicabl(aggrupation: cu.uci.dbreplication.transportable.ReplicableStructureGroup)
Descripción:	Aplica configuraciones a la agrupación y, luego de salvarla localmente, inicia su envío.

Nombre:	applyFilterToStructure (aggrupation: cu.uci.dbreplication.transportable.ReplicableStructureGroup)
Descripción:	Aplica los filtros de destinatario al ReplicableStructureGroup.
Nombre:	saveLocal (replicableStructureGroup: cu.uci.dbreplication.transportable.ReplicableStructureGroup)
Descripción:	Salva localmente los cambios en la BD embebida Berkeley.
Nombre:	sendSplittedStructureGroup (aggrupation: cu.uci.dbreplication.transportable.ReplicableStructureGroup)
Descripción:	Delimita las acciones contenidas el RSG por destinos.
Nombre:	send (replicableStructureGroup: cu.uci.dbreplication.transportable.ReplicableStructureGroup)
Descripción:	Comprueba si se está enviando, de lo contrario envía para cada uno de los objetivos.
Nombre:	getTargetsForStructureReplics(replicableStructureGroup: cu.uci.dbreplication.transportable.ReplicableStructureGroup): Set<Target>
Descripción:	Obtiene los destinos, correspondientes al RSG, hacia los cuales se efectuará la réplica.
Nombre:	monitorSend (replicableStructureGroup: cu.uci.dbreplication.transportable.ReplicableStructureGroup, targets : java.util.Set<String>)
Descripción:	Persiste el envío de los RSG hacia los objetivos especificados.
Nombre:	sendComplete (aggrupation: cu.uci.dbreplication.transportable.ReplicableStructureGroup)
Descripción:	Envía el RSG.
Nombre:	setSender(sender : cu.uci.dbreplication.sender.Sender)
Descripción:	Permite modificar el objeto Sender.
Nombre:	onIncorrectCountStructureGroupReceived (targetId: String, expectedCount: long)
Descripción:	Como ha recibido una notificación de contador incorrecto, intenta reenviar el RSG que se corresponde con ese contador esperado.
Nombre:	onDeliveryReceiverReplicableStructureGroup (targetId: String, aggrupationId: String)

Descripción:	Como el RSG ha llegado satisfactoriamente a su destino, cambia el estado de la variable sendingStructure para que se pueda enviar otro RSG e intenta enviar el próximo RSG para el destino especificado.
Nombre:	onDeliveryAplicationReplicableStructureGroup (targetId: String, aggrupationId: String)
Descripción:	Como las acciones contenidas en el RSG fueron aplicadas satisfactoriamente, elimina de la BD local la relación de ese RSG.
Nombre:	sendStoredStructureCounts()
Descripción:	Envía a través de notificaciones el valor de los contadores de RSG que faltan por enviar.
Nombre:	checkInterruptedReplicableStuctureGroupSent()
Descripción:	Chequea si ha sido interrumpido el envío de algún RSG, en caso de que existan RSG pendientes, los envía nuevamente antes de que sean enviados los nuevos RSG.

Fuente: elaboración propia

Anexo 14: Descripción de la clase ReceiverManager.

Descripción de la clase ReceiverManager	
Nombre: ReceiverManager	
Tipo de clase: Controladora	
Atributos	**Tipos**
logger	**Protected**
structureReceivedStorageDao	**Private**
registrationManager	**Private**
monitoringManager	**Private**
deliveryManager	**Private**
Responsabilidades	
Nombre:	afterPropertiesSet()
Descripción:	Método a implementar de la interfaz InitializingBean.
Nombre:	onReplicableStructureGroupArrive(replicableStructureGroup: cu.uci.dbreplication.transportable.ReplicableStructureGroup)

Descripción:	Recibe el RSG enviado por otros nodos de réplica y notifica la llegada.
Nombre:	persistInLocalDB(aggrupation: cu.uci.dbreplication.transportable.ReplicableStructureGroup)
Descripción:	Antes de aplicar los cambios contenidos en el RSG, el RSG es salvado localmente en la base de datos Berkeley.
Nombre:	onCountStructuteNotification (senderId: String, count: long)
Descripción:	Recibe notificación de contador remoto, si el contador es incorrecto, solicita el RSG con el contador correspondiente a través de una notificación.

Fuente: elaboración propia

Anexo 15: Descripción de la clase ApplicatorManager.

Descripción de la clase ApplicatorManager	
Nombre: ApplicatorManager	
Tipo de clase: Controladora	
Atributos	**Tipos**
aplicatorSchemaChange	**Private**
Responsabilidades	
Nombre:	persist(replicableStructureGroup: cu.uci.dbreplication.transportable.ReplicableStructureGroup)
Descripción:	Inicia la aplicación de cambios de esquemas contenidos en el grupo de estructura replicable.
Nombre:	execute(replicableStructureGroup: cu.uci.dbreplication.transportable.ReplicableStructureGroup)
Descripción:	Inicia la aplicación de los cambios contenidos en el grupo de estructuras replicables.

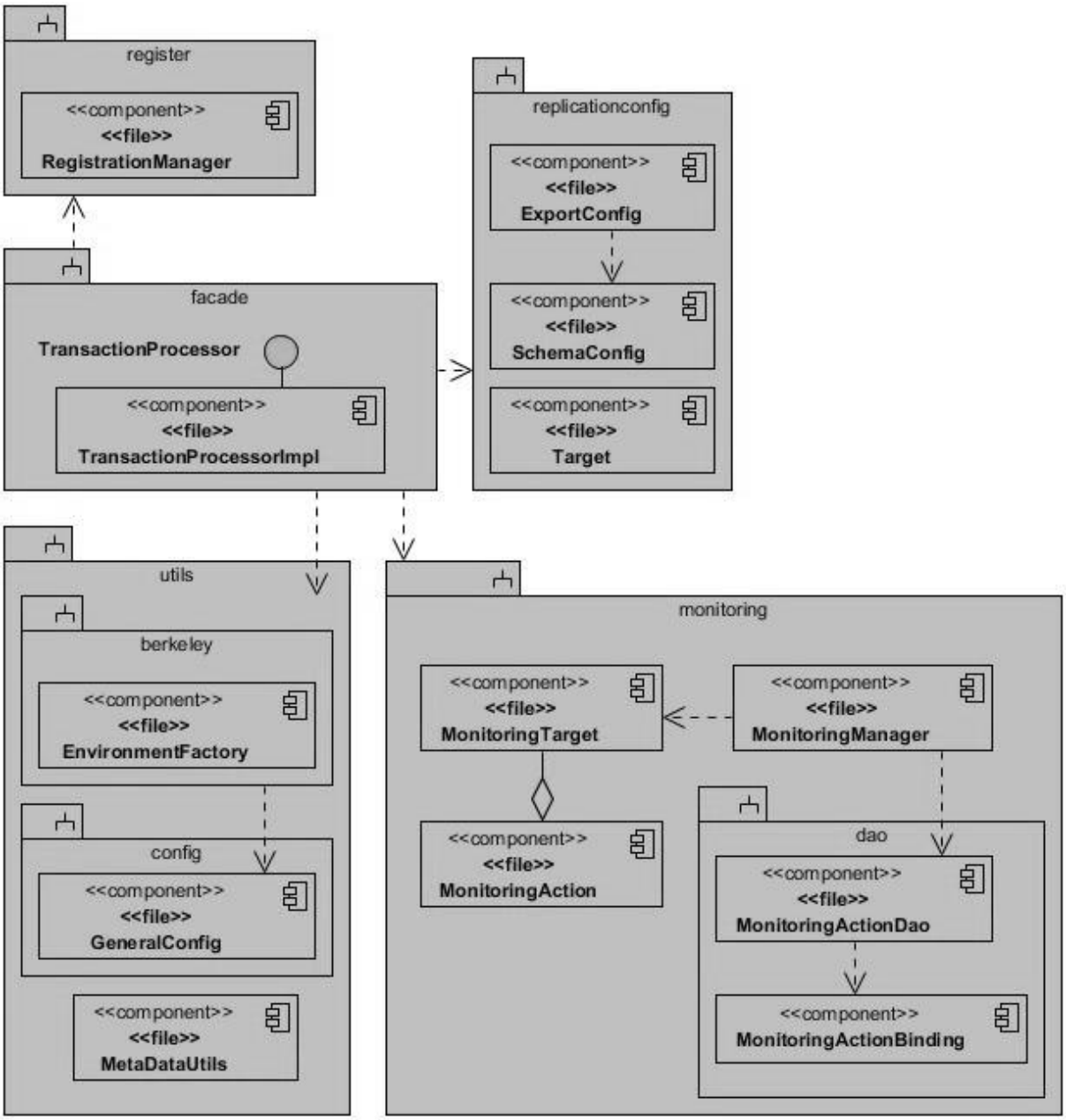
Fuente: elaboración propia

Anexo 16: Descripción de la clase DialectUtils.

Descripción de la clase DialectUtils
Nombre: DialectUtils
Tipo de clase: Controladora

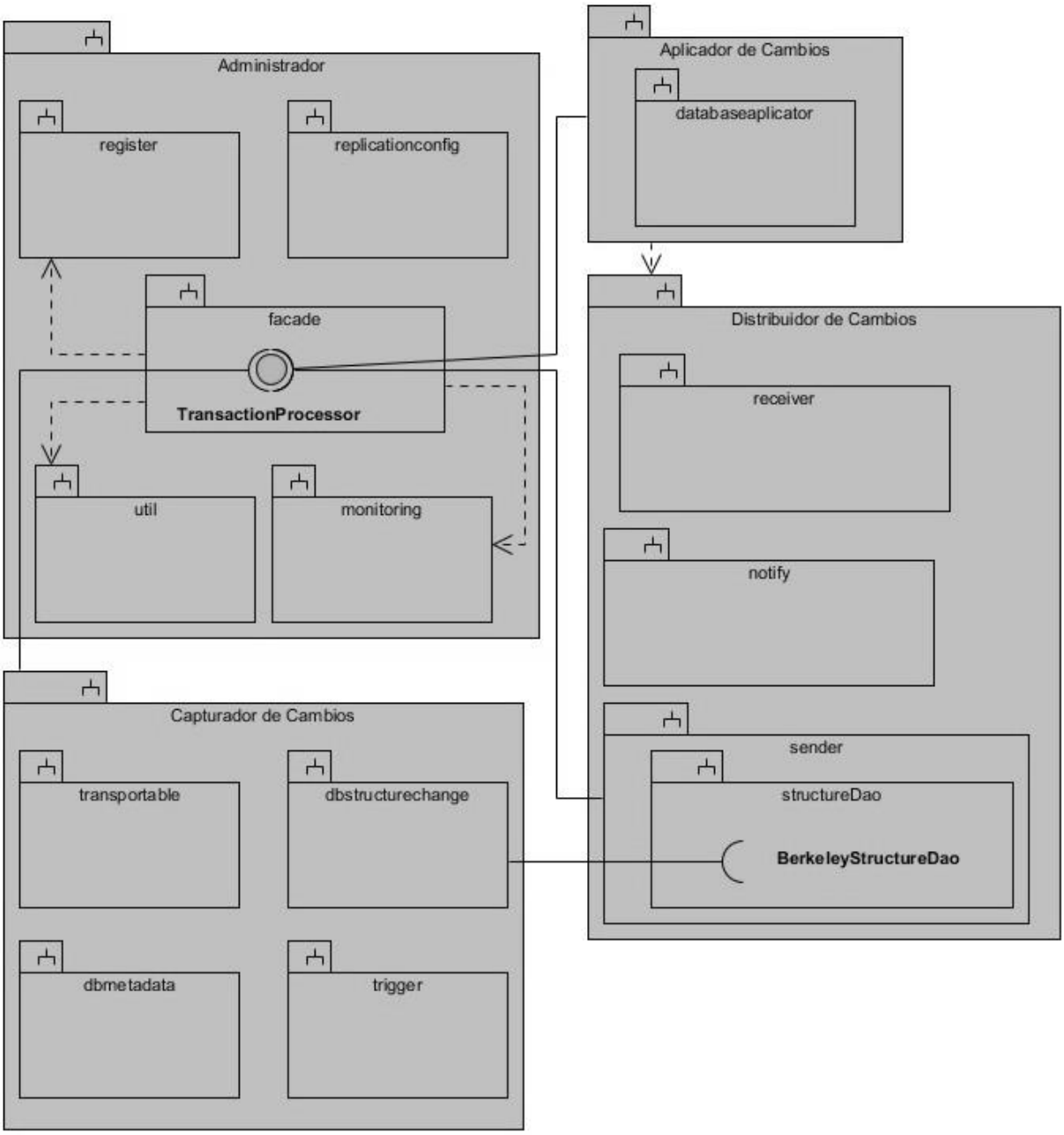
Responsabilidades	
Nombre:	getCantActionstoReplics(dialect: cu.uci.dbreplication.triggers.DBHandler.HibernateDialect): String
Descripción:	Obtiene la cadena de la consulta que selecciona la cantidad de elementos a replicar, es decir, la cantidad de tuplas contenidos en la tabla espejo.
Nombre:	getOfMirrorTable(dialect: cu.uci.dbreplication.triggers.DBHandler.HibernateDialect, cantElementToRead: int): String
Descripción:	Obtiene la cadena que selecciona de la tabla espejo una cantidad de tuplas equivalente a la cantidad que se especifica por parámetros, esta cantElementToRead es la cantidad de elementos que puede contener un RSG.
Nombre:	deleteTupleOfMirrorTable(dialect: cu.uci.dbreplication.triggers.DBHandler.HibernateDialect, id: int): String
Descripción:	Obtiene la cadena que elimina de la tabla espejo de la BD la acción con el id especificado por parámetro.
Nombre:	sentenceCreateTable(dialect: cu.uci.dbreplication.triggers.DBHandler.HibernateDialect, table: cu.uci.dbreplication.dbmetadata.Table): String
Descripción:	Obtiene la cadena que construye la sentencia “CREATE TABLE”.
Nombre:	sentenceDropTable(dialect: cu.uci.dbreplication.triggers.DBHandler.HibernateDialect, tableName: string): String
Descripción:	Obtiene la cadena que construye la sentencia “DROP TABLE”.

Anexo 17: Diagrama de componentes para el sub-sistema Administrador de cambios.



Fuente: elaboración propia.

Anexo 18: Relación de los sub-sistemas de implementación diseñados.



Fuente: elaboración propia.

Anexo 19: Estándares de codificación java.

Variables: para este identificador se empleó la variante *lowerCamelCase*. Las variables se escribirán siempre en minúsculas. Las variables compuestas tendrán la primera letra de cada palabra componente en mayúsculas.

- StringBuffer bf;

Clases:

Clases **Controladoras**: deben terminar con la palabra "**Manager** ", ejemplos:

- `public class DBStructureChangeManager {}`
- `public class MetadataManager {}`
- `public class AplicatorManager {}`
- `public class SenderManager {}`

Clases **Exceptions:** estas clases deben terminar con la palabra "**Exception**", ejemplo:

- `public class DataAccessException extends Exception{}`

Clases **DAO**: las clases que se utilicen como Objeto de Acceso a Datos deben terminar con la palabra "**Dao**", ejemplos:

- `public interface DBStructureDao {}`
- `public interface NotificationDao {}`
- `public interface SenderStorageDao {}`

Anexo 20: Instrumento de evaluación de la métrica TOC.

Clases	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
Table	36	Baja	Baja	Alta
Column	46	Baja	Baja	Alta
Change	10	Baja	Baja	Alta
MetadataManager	35	Baja	Baja	Alta
DBStructureChangeManager	46	Baja	Baja	Alta
DialectUtils	134	Alta	Alta	Baja
HibernateDialect	159	Alta	Alta	Baja
DBReadLogChange	3	Baja	Baja	Alta
MySqlServerDialect	185	Alta	Alta	Baja
SqlServerDialect	180	Alta	Alta	Baja
ReplicableStructureGroup	19	Baja	Baja	Alta
AplicatorManager	11	Baja	Baja	Alta
ReplicableStructureAction	25	Baja	Baja	Alta

ActionType	0	Baja	Baja	Alta
Promedio de métodos por clases	**63,5**			

Fuente: elaboración propia

Resumen de la aplicación de la métrica TOC

Indicadores de Calidad	**Evaluación %**		
	Baja	**Media**	**Alta**
Responsabilidad	72	0	28
Complejidad de implementación	72	0	28
Reutilización	28	0	72

Fuente: elaboración propia

Anexo 21: Instrumento de evaluación de la métrica RC.

Clases	**Relaciones**	**Acoplamiento**	**Complejidad M.**	**Reutilización**	**Cantidad de Pruebas**
DBStructureChangeManager	4	Alta	Alta	Baja	Alta
Column	1	Baja	Baja	Alta	Baja
MetadataManager	1	Baja	Baja	Alta	Baja
Table	1	Baja	Baja	Alta	Baja
Change	2	Media	Media	Media	Media
DBReadLogChange	1	Baja	Baja	Alta	Baja
DialectUtils	1	Baja	Baja	Alta	Baja
HibernateDialect	1	Baja	Baja	Alta	Baja
MySqlServerDialect	1	Baja	Baja	Alta	Baja
SqlServerDialect	1	Baja	Baja	Alta	Baja
ReplicableStructureGroup	1	Baja	Baja	Alta	Baja
AplicatorManager	1	Baja	Baja	Alta	Baja
ReplicableStructureAction	1	Baja	Baja	Alta	Baja

ActionType	1	Baja	Baja	Alta	Baja
Promedio	**1,28**				

Fuente: elaboración propia

Resumen de la aplicación de la métrica RC

Indicadores de Calidad	**Evaluación %**		
	Baja	**Media**	**Alta**
Acoplamiento	86	7	7
Complejidad de mantenimiento	86	7	7
Reutilización	7	7	86
Cantidad de Pruebas	86	7	7

Fuente: elaboración propia