Universidad de las Ciencias Informáticas

Facultad 4

Centro de Consultoría y Desarrollo de Arquitecturas Empresariales

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

Título: Proceso de réplica de estructura entre bases de datos gestionadas en Oracle para el Replicador de datos Reko.

Autor: Norberto Williams Barrientos Trenal.

Tutoras: Ing. Liset Martínez Almaguer.

Ing. Yanet Riquelme Santiago.

CDAE

”La Habana, 2017”

"El futuro de nuestra patria tiene que ser necesariamente un futuro de hombres de ciencia, tiene que ser un futuro de hombres de pensamiento, porque precisamente es lo que más estamos sembrando; o que más estamos sembrando son oportunidades a la inteligencia (...)"

AGRADECIMIENTO

A mi mamá por todo su esfuerzo y cariño, por ayudarme y apoyarme en mis estudios y en todo lo que he necesitado, por su firmeza. Porque con su ejemplo de esfuerzo y perseverancia me ha inspirado a superarme cada día más, por ser la persona que más he admirado y ser un pilar importante en mi vida.

A mi padre por apoyarme en todo, por sus consejos, por ser una persona que he querido y admirado desde mis primeros pasos.

A mi hermana, te amo, porque eres mi confidente, mi mejor amiga, por ser incondicional.

A mi padrastro, por ser una persona con muchos conceptos y valores, por sus concejos, por su ayuda incondicional, por su amor a mi madre, te admiro.

A mis tutoras Liset y Yanet por su confianza, su apoyo en todo momento, por ser las personas que me han guiado en el transcurso de mi tesis, por estar siempre disponibles, les agradezco mucho, gracias por todo.

A mis amigos por los momentos que hemos compartido juntos, por sus consejos, por sus risas, sus chistes, por estar cada vez que los he necesitado y porque de cada uno de ellos he aprendido algo.

Y a Dios, por darme la oportunidad de tenerlos a ustedes y de haberlos conocido.

DEDICATORIA

A mí mamá, mí papa , mí hermana y padrastro por ser las personas más importantes en mi vida, porque sin ustedes no sería la persona que soy, los amo.

DECLARACIÓN JURADA DE AUTORÍA

Declaramos ser autores de la 0presente 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 2017.

Firma del autor:

_____

Norberto Williams Barrientos Trenal.

Firma de las tutoras:

_____ _____

Ing. Liset Martínez Almaguer. Ing. Yanet Riquelme Santiago.

RESUMEN

El siguiente trabajo de diploma expone una solución para el proceso de réplica de estructura entre bases de datos gestionadas en *Oracle* para el Replicador de datos Reko, el cual posibilita la actualización automática de los cambios ocasionados en la estructura de base de datos *Oracle*, por la realización de consultas *DDL* (Lenguaje de Definición de Datos, por sus siglas en inglés).

El proceso desarrollado permite la aplicación de los cambios de estructura capturados en la base de datos local hacia las bases de datos distribuida. Para esto, cuenta con procesos que aseguran la captura de los cambios de estructura, la construcción de las acciones que contienen los cambios, el envío de las acciones empleando el servidor de mensajería *ActiveMQ*, y la aplicación de los cambios en los nodos remotos, utilizando para ello el dialecto de *Oracle*.

EL trabajo investigativo describe: la problemática inicial que dio comienzo a la extensión del proceso de réplica de estructura en *Oracle* para el *software* Reko. El manejo de métodos teóricos y empíricos, la comparación con sistemas homólogos a nivel mundial con características similares arrojando como resultado la creación de este nuevo proceso. Se empleó la metodología AUP en su variante UCI para describir el diseño e implementación del proceso apoyándose en los artefactos que propone. Por último, se termina la extensión del proceso el cual fue elaborado completamente con herramientas *Open Source* y librerías de clases con licencias gratuitas y la realización de las pruebas correspondientes a las funcionalidades requeridas.

Palabras claves: base de datos, proceso, réplica de estructura, *software*.

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

En la actualidad existe un gran aumento de la informatización de procesos, esto respalda la descentralización geográfica de las organizaciones y entidades, lo que ha impulsado el empleo de los Sistemas de Bases de Datos Distribuidas (*SBDD*). Los *SBDD* son un tipo de *software* muy específico, dedicado a servir de interfaz entre la Base de Datos (*BD*), el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, manipulación de datos y consulta. Se encargan de mantener la validez y la consistencia de los datos almacenados, facilita que el acceso y procesamiento a los datos sea más rápido.(1)

Con el empleo de los *SBDD* se ha logrado un avance significativo para el almacenamiento de la información. La posibilidad de compartir datos, eliminar inconsistencias, mejorar la seguridad; y como valor agregado, recolectar cualquier tipo de información, ahorrar espacio de almacenamiento y replicar datos ha convertido a los *SBDD* en la principal fuente de almacenamiento en el campo informático.

Los Sistemas Distribuidos necesitan un mecanismo que les facilite la gestión de la información entre las diferentes *BD*, por ende utilizan replicadores de datos. Los replicadores de datos facilitan la recuperación, amplían la tolerancia a fallos y fiabilidad en la distribución de los datos.(2)

En nuestro país, 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. Este replicador 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, protección y recuperación, así como satisfacer necesidades relacionadas con la distribución de datos entre los gestores más populares. Constituye un *software* maduro y una solución robusta ante un amplio conjunto de escenarios de réplica para los cuales se ha probado. Su implementación se realizó con herramientas *Open Source*[1] y bibliotecas de clases con licencia gratuita, constituye una solución soberana de alto impacto en la sustitución de importaciones como una alternativa construida sobre una arquitectura en tecnologías libres.(3)

El *software* Replicador de datos Reko brinda soporte de réplica de datos a gestores de *BD* populares como: *PostgreSQL*, *Oracle*, *MySQL* y *Microsoft SQL Server*. Cuenta con un módulo

[1] De código abierto, término con el que se conoce al software distribuido y desarrollado libremente. La idea bajo el concepto de código abierto es la siguiente: cuando los programadores pueden leer, modificar y redistribuir el código fuente de un programa, éste evoluciona, se desarrolla y mejora.

de Configuración de Réplica, el cual le permite al usuario crear las configuraciones de réplica de estructura, este proceso únicamente se realiza para el gestor *PostgreSQL*.

El módulo de Configuración de Réplica le brinda la posibilidad al usuario mediante interfaces gráficas elegir el esquema que se desea replicar, así como los diferentes tipos de acciones (replicar la creación de un nuevo esquema, replicar la modificación del nombre de algún esquema y replicar la eliminación de algún esquema).

Cuando se trabaja con el gestor de *BD Oracle* y se aplican acciones de Lenguaje de Definición de Datos (*DDL*[2], por sus siglas en inglés) sobre la estructura en la *BD* se necesita reiniciar la aplicación, ya que dichas acciones no son reconocidas automáticamente en la configuración de réplica de datos, siendo el cambio invisible en la configuración de réplica de estructura. Como no actualiza inmediatamente los cambios realizados sobre la configuración de réplica, provoca que disminuya el aprovechamiento laboral del personal involucrado en este proceso debido que deben solucionar los problemas de forma manual. Posibilitando que se introduzcan errores en la estructura de la *BD*, ocasionando inconsistencias entre los datos que se replican, tablas o datos incompletos e incluso problemas de integridad referencial, todo esto afecta la coherencia de los datos.

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

¿Cómo actualizar automáticamente los cambios realizados sobre la estructura de una *BD Oracle* para garantizar la coherencia en los datos que se replican con 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 distribuidos.

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

Desarrollar un proceso para la réplica de estructura en el gestor de BD *Oracle*, que permita actualizar automáticamente los cambios realizados sobre la estructura para garantizar la coherencia en los datos que se replican con el Replicador de datos Reko.

Por lo que el **campo de acción** comprende el proceso de captura y aplicación de cambios de estructura en *BD Oracle*.

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

[2] Lenguaje de Definición de Datos

- Elaboración del marco teórico de la investigación a partir del estado del arte sobre las tendencias actuales en la réplica de base de datos y la evolución de las herramientas en diferentes gestores.
- Aplicación de la metodología para definir los métodos y técnicas necesarias que guiarán el desarrollo del proceso.
- Definición del ambiente de desarrollo a partir de las herramientas, tecnologías y lenguajes a utilizar.
- Identificación de las capacidades que tiene que ser alcanzadas por el sistema para cumplir los objetivos trazados, a partir de los requisitos funcionales y no funcionales.
- Diseño del proceso para detallar las historias de usuarios, diagramas de clases y diagramas de componentes, con sus respectivas descripciones.
- Extender del proceso de réplica de estructura para el gestor de *BD Oracle* del *software* Replicador de datos Reko.
- Realización de pruebas al proceso en entornos de producción para valorar la calidad del producto implementado.
- Valoración de los resultados obtenidos.

La **idea a defender** en la presente investigación, se plantea de la siguiente manera:

Con el uso del proceso desarrollado para la réplica de estructura en el gestor de *BD Oracle*, se actualizará automáticamente los cambios realizados sobre las estructuras que garantizará la coherencia en los datos que se replican con el Replicador de datos Reko.

En la confección de la investigación se utilizaron **métodos científicos**, todos bajo la idea dialéctica-materialista como método general.

Los **métodos teóricos** empleados se exponen a continuación:

- **Histórico - Lógico**: hizo posible la determinación de los antecedentes en su devenir histórico, tendencias y regularidades del objeto de estudio y campo de acción, así como un estudio cronológico sobre las herramientas de réplica que realizan trabajos similares, donde se obtuvieron conocimientos que fueron aplicados en la solución del problema presentado.
- **Analítico – Sintético**: permitió la determinación de las generalidades y especificidades en el objeto de estudio y el campo de acción, así como en la fundamentación teórica y elaboración del proceso de réplica de estructuras en el gestor de *BD Oracle* con el Replicador de datos Reko.
- **Modelación**: permitió generar los artefactos necesarios y diseñar el proceso para la réplica de estructura en el gestor de *BD Oracle*.

Se utilizaron como **métodos empíricos:**

- **Observación científica**: mediante su aplicación se detectaron las necesidades existentes y posibles mejoras a incorporar al *software*, con el propósito de extender sus funcionalidades y alcanzar un producto de mayor calidad.
- **Consulta de la información en todo tipo de base**: permitió la elaboración del marco teórico de la investigación, en sus aspectos conceptuales y metodológicos.
- **Nivel estadístico**: mediante la estadística descriptiva, con el empleo de gráficos y tablas, hizo posible la organización y regularización de la información obtenida.
- **Pruebas de validación**: permitió detectar y corregir los defectos, disminuir los riesgos, mejorar la calidad y fiabilidad del *software* y las desviaciones respecto al objetivo fijado.

Con la presente investigación se espera como resultado que el Replicador de datos Reko cuente con un proceso para la réplica de estructura en el gestor de *BD Oracle*, que permita actualizar automáticamente los cambios realizados y garantizar la coherencia en los datos que se replican, proponiendo así una solución más completa ante las necesidades de réplica.

El trabajo de diploma consta de introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas y anexos.

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 aplican pruebas de diferentes tipos al proceso para demostrar su robustez.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

El presente capítulo proyecta los principales conceptos relacionados con la problemática a resolver. Expone los conceptos asociados con el proceso de réplica de estructura y las soluciones de réplica. Además, se realiza un estudio científico del tema para conformar el basamento teórico utilizado en la resolución del problema científico y en él se define la metodología, herramientas y lenguaje a utilizar en la solución propuesta.

1.1 Marco conceptual

El marco conceptual es un intento por caracterizar todos aquellos elementos que intervienen en el proceso de la investigación. A través de la revisión de publicaciones de varios autores y teorías se busca poder encontrar aquellas definiciones, conceptos y líneas para enmarcar la investigación e interpretar los resultados y las conclusiones que se alcanzan.

A lo largo de esta sección se espera poder dar al lector una mejor comprensión y entendimiento de la investigación desarrollada.

1.1.1 Base de datos

El autor Damián Pérez Valdés[3] plantea:
"Una base de datos es un "almacén" que nos permite guardar grandes cantidades de información de forma organizada para que luego podamos encontrar y utilizar fácilmente".(4)

Según Michael V. Mannino[4]:
"…*una base de datos es una colección de datos persistentes que pueden compartirse e interrelacionarse*".(5)

La Dra. María del Carmen Gómez Fuentes[5] expone:
"…una base de datos no es más que un conjunto de información (un conjunto de datos) relacionada que se encuentra agrupada o estructurada." (6)

El autor de la tesis considera validos los conceptos antes planteados y acordes a los objetivos de la investigación. A partir de los cuales considera que:

Una *BD* es una colección de datos organizados, persistentes y relacionados entre sí, que se encuentran agrupados o estructurados, los cuales se pueden encontrar y utilizar fácilmente.

[3] Damián Pérez Valdés, Web master, administrador de sistemas, con experiencia en desarrollo web y de aplicaciones.
[4] Michael V. Mannino, profesor asociado de sistemas de información, Co-Director del Doctorado del *CSIS*, de la escuela de negocios Universidad de Colorado Denver.
[5] Dra. María del Carmen Gómez Fuentes, Departamento de Matemáticas Aplicadas y Sistemas. División de Ciencias Naturales e Ingeniería Universidad Autónoma Metropolitana, Unidad Cuajima, México.

1.1.2 Base de datos distribuida

El autor Alejandro Gutiérrez Díaz[6] emite que:
"*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*." (7)

Se asume dicho concepto planteado por el autor citado anteriormente porque se considera válido y acorde a los objetivos de investigación.

En las *BDD*, las *BD* se asignan en forma de nodos como se expone en la Figura 1, donde los nodos pueden ser adicionados fácilmente, los cuales poseen la capacidad de procesamiento autónomo e indica que pueden efectuar operaciones locales o distribuidas. La probabilidad de que un nodo afecte el funcionamiento del sistema es baja.

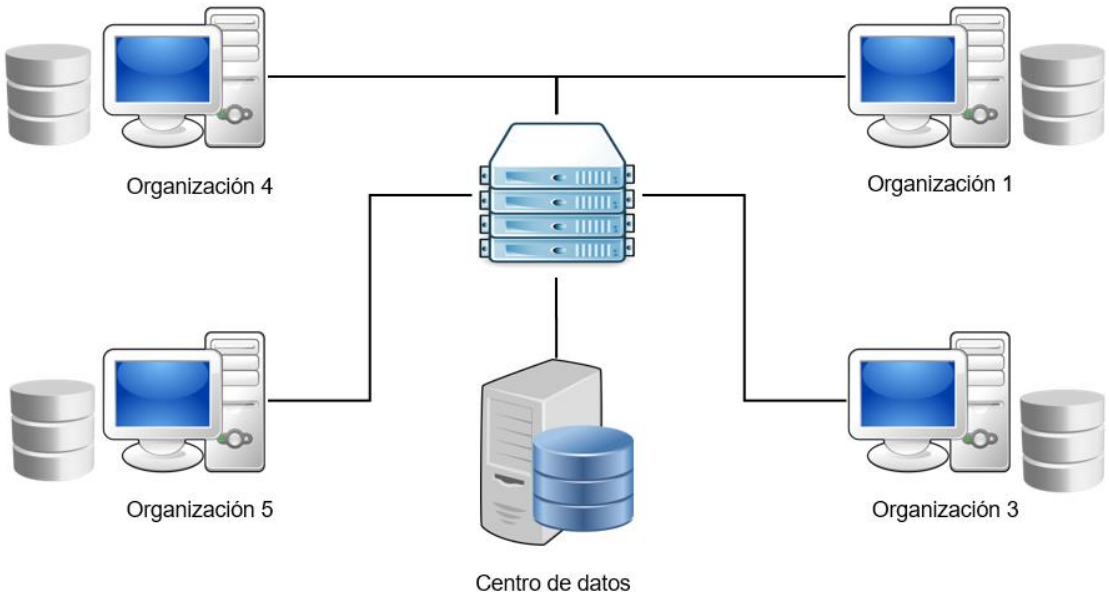


Figura 1. Interconexión de una BDD.
Fuente: elaboración propia.

1.1.3 Esquema o estructura de una BD

José Evaristo Pacheco Velasco[7] expone:
"*El esquema de una base de datos (en inglés, Database Schema) describe la estructura de una base de datos, en un lenguaje formal soportado por un Sistema Administrador de Base de Datos (DBMS). En una Base de datos Relacional, el esquema define sus tablas, sus*

[6] Alejandro Gutiérrez Díaz, profesor, investigador y Master en Ciencias.
[7] José Evaristo Pacheco Velasco, master en ciencias y miembro del Instituto Tecnológico de Veracruz.

campos en cada tabla y las relaciones entre cada campo y cada tabla. *Aunque generalmente el esquema es definido en un lenguaje de Base de datos, el término se usa a menudo para referirse a una representación gráfica de la estructura de base de datos (Diseño de lógico de la base de datos)*"(8)

El concepto planteado por el autor se considera válido y acorde a los objetivos de la investigación, se concreta que un esquema, define tablas, sus campos en cada tabla y las relaciones entre cada campo y cada tabla, refiriéndose al diseño lógico de la *BD*.

1.1.4 Coherencia de los datos

José Antonio Ocampo[8] declara que la coherencia de los datos:
"…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 diversas naturalezas".(9)

El colectivo de la especialidad de *BD* del Instituto Nacional de Estadística de España emite que:
"…es la idoneidad de los datos para ser combinados de forma fiable de diferentes maneras y para distintos usos".(10)

Se toma las opiniones de los autores anteriormente citados, y se asume que la coherencia de los datos en el entorno de réplica es la idoneidad de los datos al ser enviados desde un nodo origen hacia un nodo destino.

1.1.5 Réplica

Según el sitio oficial de Oracle[9] réplica es:
"... el proceso de copiar y mantener objetos en varias bases de datos que componen un sistema de base de datos distribuida. La replicación puede mejorar el rendimiento y proteger la disponibilidad de aplicaciones. Por ejemplo, una aplicación podría normalmente acceder a una base de datos local en lugar de un servidor remoto para minimizar el tráfico de red y lograr el máximo rendimiento. Además, la aplicación puede seguir funcionando si el servidor local experimenta un fracaso porque otros servidores con datos replicados siguen estando accesibles."(11)

El autor Omar Chavez[10] declara:

[8] José Antonio Ocampo, Secretario General de las Naciones Unidas para asuntos económicos y sociales, y ex secretario ejecutivo de la *CEPAL*.
[9] Potente herramienta cliente/servidor para la gestión de *BD*.
[10] Omar Chavez, cibernético, Máster en el soporte de *BD*.

"…es el proceso de copiar y administrar objetos de base de datos, tales como tablas, hacia múltiples bases de datos en localidades remotas que son parte de un sistema de bases de datos distribuido. Los cambios ejecutados en una localidad son capturados y guardados localmente antes de ser aplicados a las localidades remotas."(12)

Para el autor Randy Urbano[11]:
"…*la réplica se entiende como el proceso de copiar y mantener objetos de una base de datos, entiéndase tablas, triggers[12], procedimientos, funciones, índices, etcétera, en múltiples bases de datos, de tal forma que los cambios realizados localmente, sean enviados a las bases de datos remotas y luego sean aplicados*".(13)

Vivian Romero Buchillón[13] expresa:
"es un mecanismo utilizado para propagar y diseminar datos en un ambiente distribuido, con el objetivo de tener mejor confiabilidad y performance de una BDD, mediante la reducción de dependencia de un sistema de base de datos centralizado".(14)

En correspondencia con los conceptos planteados anteriormente el autor de la tesis, asume que la réplica es el proceso de copiar y mantener objetos en varias *BD* que componen un *SBDD*, de tal forma que los cambios realizados en la *BD* local, sean enviados y aplicados en las *BD* remotas, garantizando la confiabilidad y coherencia de los datos.

Lenguaje de Manipulación de Datos

Un Lenguaje de Manipulación de Datos (*DML*[14] en inglés) es un lenguaje proporcionado por el sistema de gestión de *BD* que permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.(15)

Los *DML* se clasifican en dos grandes grupos:

- **lenguajes de consulta procedimentales**: en los lenguajes procedimentales el usuario da instrucciones al sistema para que realice una serie de procedimientos u operaciones en la *BD* para calcular un resultado final.
- **lenguajes de consulta no procedimentales**: en los lenguajes no procedimentales el usuario describe la información deseada sin un procedimiento específico para obtener esa información.(15)

[11] Randy Urbano, escritor técnico de *Oracle*.
[12] Conocido como disparador. Es un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación *DML*.
[13] Vivian Romero Buchillón, de origen cubano su especialidad es Analista de Sistemas, Programación.
[14] Lenguaje de Manipulación de Datos.

Los comandos *DML* son:

- ***INSERT***: una sentencia *INSERT* de *SQL*[15] agrega uno o más registros a una tabla en una *BD* relacional.
- ***UPDATE***: una sentencia *UPDATE* de *SQL* es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.
- ***SELECT***: una sentencia *SELECT* de *SQL* es utilizada para consultar registros de la *BD* que satisfagan un criterio determinado.
- ***DELETE***: una sentencia *DELETE* de *SQL* borra uno o más registros existentes en una tabla.(15)

1.1.6 Lenguaje de Definición de Datos

Un *DDL* es un lenguaje proporcionado por el sistema de gestión de *BD* que permite a los usuarios de la misma llevar a cabo las tareas de definición de las estructuras que almacenarán los datos, así como de los procedimientos o funciones que permitan consultarlos.(16)

Los comandos *DDL* son:

- ***CREATE***: utilizado para la creación de nuevas estructuras.
- ***DROP***: empleado para eliminar estructuras.
- ***ALTER***: utilizado para modificar las estructuras.(16)

1.1.7 Proceso

Para el Dr. José María Paganini[16] un proceso se define:
"*Como una serie de actividades interrelacionadas que presentan una relación lógica entre sí, para obtener un resultado esperado*."(17)

Se puede concluir que un proceso es una secuencia de pasos dispuesta con algún tipo de lógica que se enfoca en lograr algún resultado específico.

1.2 Replicadores

El aumento a la informatización de las empresas genera constantemente un gran número de información. Para mantener actualizadas las *BD* geográficamente distribuidas las empresas comercializan soluciones de réplica. Algunas incluyen la solución de réplica en el *software* y otras como soluciones que se pueden adquirir de manera independiente. Se realizó un estudio

[15] Lenguaje de consulta estructurado (*SQL*) es un lenguaje de *BD* normalizado, utilizado por el motor de *BD* de *Microsoft Jet*.
[16] Miembro Correspondiente Nacional de la Academia Nacional de Medicina en Argentina.

de replicadores para conocer como realizan el proceso de captura y aplicación de los cambios de estructura, en busca de una propuesta para el proceso de réplica de estructura en el gestor de *BD Oracle* con el Replicador de datos Reko.

1.2.1 DBMoto Cloud Edition

DBMoto Cloud Edition es una herramienta tecnológica que ofrece una solución para ejecutar operaciones de captura de datos entre *BD* sin conexión directa a la red, en lugar de un *web service*. Para un óptimo rendimiento, el producto es diseñado para capturar cambios en la *BD* origen y propagar esos cambios al o los destinos. Inicialmente, es necesario generar una copia completa de las tablas envueltas y actualizar la *BD* destino con un completo conjunto de datos capturados.(18)

DBMoto Cloud Edition provee la replicación y transformación de datos en tiempo real. Incorpora tecnología de acceso para garantizar un alto rendimiento y un óptimo nivel de seguridad en la replicación de sus datos, con el valor agregado de estar disponible para transmitir datos relacionales desde el origen al destino utilizando la nube (*cloud*[17]). Captura de datos, el cual reduce significativamente la cantidad de movimientos de datos para actualización y puede ser usada para actualizar datos remotos atrás de *firewalls*[18].(18)

Fue especialmente diseñado para replicar y capturar datos en modo "*Refresh*" y "*Mirroring*".

- Modo ***Refresh*** (*Snapshot*[19]): lee los datos, aplica las reglas de mapeo definidas por el administrador y escribe los resultados en la *BD* de destino.
- Modo ***Mirroring*** (Captura de datos): replicación desde el origen al destino en tiempo real, basado en los *log/journal*[20] transaccionales de las *BD* y aplicando captura de datos para minimizar el tráfico de datos.(18)

Con DBMoto, los usuarios pueden definir las replicaciones y transformaciones de los datos aplicando reglas de negocios y mapeo. Las reglas de negocio pueden ser aplicadas a través de *scripts*[21], los cuales son usados para filtrar datos o para agregar lógica de negocios a las replicaciones. Estos son generalmente implementados a través de una generación automática de eventos durante la replicación con DBMoto. Los *scripts* pueden configurar las vistas de las

[17] Tecnología orientada al uso de equipos pequeños y portátiles (que utilizan servicios *online*), en la cual todos los servicios prestados al ordenador se hacen directamente desde Internet, con la cual se simplifica la instalación de *software* y se optimiza el uso del espacio del disco duro.
[18] Sistema de defensa basado en el hecho de que todo el tráfico de entrada o salida a la red debe pasar obligatoriamente por un sistema de seguridad capaz de autorizar, denegar, y tomar nota de todo aquello que ocurre, de acuerdo con una política de control de acceso entre redes.
[19] Copia instantánea de volumen.
[20] Mecanismo por el cual un sistema informático puede implementar transacciones.
[21] 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.

tablas de origen y destino, añadir actividad a los *logs* y actualizar las tablas de destino. También incluye un poderoso generador de expresiones para la transformación de datos.

Algunas de sus características son**:**

- Soporta gestores de *BD* como: *Oracle*, *Microsoft SQL Server*, *MySQL*, *Server Enterprise*, *PostgreSQL*, y *MS Access*.
- Modos de replicación y captura de datos: *Refresh* y *Mirroring*.
- Replicación de datos desde origen al destino, usando la nube (*cloud*) y *web service* para transferencia de datos.
- Soporte para múltiples *BD* (origen y destino).
- No requiere programación en las plataformas de *BD* de origen y destino.
- Habilidad para leer *logs* de transacciones de *BD* (para hacer más eficiente el acceso a los datos).
- Completo *log* de reportes y accesibilidad.
- Poderosa herramienta visual con información sobre el estatus de la replicación.
- Creación automática de tablas de destino.(18)

1.2.2 SymmetricDS

SymmetricDS es un *software* de replicación de datos asíncrona[22] que permite subscriptores múltiples y sincronización bidireccional. Utiliza tecnologías *web* y de *BD* para replicar tablas entre *BD* relacionales, casi en tiempo real. El *software* fue diseñado para escalar a un gran número de *BD*, trabajar con conexiones de bajo ancho de banda y resistir a periodos de inoperatividad de la red.(19)

Funciona con la mayoría de sistemas operativos, sistemas de archivos y *BD*, soporta un gran número de gestores de *BD* ejemplos de ellos son *MySQL*, *Oracle*, *SQL Server*, *PostgreSQL*.(20)

Una única instalación de SymmetricDS se denomina un Nodo. Un Nodo es inicializado mediante un fichero *properties* y es configurado insertando datos de configuración en una serie de tablas de *BD*. A continuación, el Nodo crea *triggers* de *BD* en las tablas de aplicación especificadas, de modo que los eventos de *BD* son capturados para ser entregados a otros Nodos SymmetricDS.(19)

[22] 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.

Algunas de sus características son:

- Licencia de código abierto *GPL*, multiplataforma.
- Pensado para tener nodos desconectados por largos periodos de tiempo.
- Es capaz de replicar los cambios estructurales de la *BD* que puedan surgir.
- Flexibilidad a la hora de declarar las reglas de replicación de los datos.
- Facilidad de configuración, basado en *triggers*.
- Alto rendimiento en ambientes con bajo ancho de banda y problemas de conexión.(19)

Replicar el esquema de *BD* en los nodos remotos y sincronizar los cambios de esquema para las tablas cuando se alteran es posible con SymmetricDS utilizando un par de técnicas simples y mediante *triggers*. En primer lugar, las tablas se pueden crear y rellenar con una carga inicial de datos cuando el nodo se registra. En segundo lugar, las definiciones de tabla se pueden enviar a los nodos necesarios para crear o modificar las tablas usando el *DDL* de la *BD*.(21)

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

- **Modificación de la Base de datos**: se ejecuta el lenguaje de definición de datos *DDL* en la base central de datos. Esto debe hacerse durante un período razonablemente sin contención. crea, elimina y altera las tablas, pero no actualiza ningún dato.
- **Sincronización con *triggers***: se ejecuta el "***sync-triggers***", proceso en SymmetricDS para detectar el cambio de esquema. Se vuelven a crear los factores desencadenantes para que coincidan con las definiciones de la tabla. Una manera fácil de hacer esto es reiniciar el servicio, o ejecutando el siguiente comando.
- **Enviar esquema**: se ejecuta el comando "***send-symadmin***" con esquema de subcomando para enviar los cambios de esquemas a nodos remotos. Este comando detecta qué cambios hacer a la tabla a través de plataformas de *BD* diferentes. Si conoce el *SQL* específico que debe ejecutar para su *BD*, puede ejecutar el comando "***send-symadmin***" con el subcomando "***send-sql***" para enviar el cambio *SQL* a nodos remotos. Este subcomando admite el envío de un nodo o grupos de nodos. Los cambios de esquema se ponen en cola, en captura de datos modificados, por lo que se reproducirán en el mismo orden en otros nodos. Este paso es alternativo.(21)

1.2.3 Oracle Streams

Oracle Streams basa su funcionamiento en mensajes, unidad básica de la información que comparte. La unidad básica para capturar los cambios se denomina *LCR*[23] de la cual existen

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

dos tipos: *row LCR* relativa a los cambios por operaciones *DML* y *DDL LCR* relativa a los de operaciones de *DDL*, como su nombre lo indica.(22)

Los mensajes pueden ser originados, transformados y/o enviados por el gestor de *Oracle* u otras herramientas externas que interactúen con Oracle Streams, lo que brinda una alta flexibilidad y variadas funcionalidades. El acceso a las bases de datos diferentes es trasparente al usuario.(22)

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*. Los datos y los eventos pueden ser cambiados o formateados por un conjunto predefinido de reglas antes de ser empaquetados en un *LCR*.
- **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.(22)

Las reglas declarativas abarcan transformación para *row LCR*, incluyendo el cambio de nombre de un esquema, cambiar el nombre de una tabla, agregar una columna, cambiar el nombre de una columna y supresión de una columna. Oracle Streams realiza transformaciones declarativas internamente, sin necesidad de invocar *PL*/*SQL*[24].

Las reglas personalizadas requieren una función definida por el usuario *PL*/*SQL* para realizar la transformación que es invocada por Oracle Streams. Una regla personalizada basada en reglas de transformación puede modificar cualquiera de *LCR* o mensajes de usuario como el tipo de datos de una columna concreta de una *LCR*. (22)

1.2.4 Replicador de datos Reko

Es un *software* de réplica de datos entre *BD*. Fue diseñado e implementado completamente usando herramientas *Open Source* 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

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

a un servidor central *JMS*[25] o en caso de un *clúster*[26] de servidores *JMS*. Con el empleo de herramientas libres y la metodología de desarrollo AUP.(23)

Su desarrollo sobre la plataforma *JEE*[27] 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*[28] *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.(23)

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 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.(2)

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 SSL29.
- Interfaz de administración de réplica.(23)

1.2.5 Resultados de la investigación

Al finalizar el estudio de las soluciones de replicación se realizó una comparación entre las herramientas, como se muestra en la tabla 1.

[25] Servicio de Mensajería de Java, del inglés *Java Message Service.*
[26] 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.
[27] Plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java, del inglés *Java Enterprise Edition.*
[28] Arquitectura reusable que brinda la estructura genérica y de comportamiento para un grupo de abstracciones de software.
[29] Capa de conexión segura, del inglés *Secure Sockets Layer.*

En la investigación realizada anteriormente se concluye que las soluciones no cumplen con los requerimientos básicos trazados, puesto que:

- Las herramientas libres como DBMoto Cloud Edition, por sí solas no cuentan con un proceso o solución que pueda ser integrado al Replicador de datos Reko para resolver las réplicas de estructura en el gestor de bases de datos *Oracle*.
- SymmetricDS está limitado en cuanto a usabilidad, debido a que las configuraciones se realizan mediante comandos en una consola de administración, lo que implica que los usuarios deban poseer un conocimiento avanzado sobre el tema.
- No se posee el código de la implementación en las fuentes consultadas.
- En ciertos casos se hace alusión a la realización de los procesos, sin ofrecer detalles en cuanto a su funcionamiento interno, ni de cómo se realiza.
- La integración de algunos de los replicadores que posean esta funcionalidad en el Replicador de datos Reko reduciría considerablemente el rendimiento del sistema.
- En las soluciones privativas no se esclarece totalmente como se realiza dicho proceso y al contar con una solución se deben invertir grandes sumas de dinero en pago de licencias del *software*.

Al detectarse las limitantes antes expuestas se hace necesario dedicar esfuerzo en el desarrollo de una solución para el Replicador de datos Reko, que independice al país de soluciones propietarias y contribuya a la independencia tecnológica.

Puede ser usado el funcionamiento de las soluciones estudiadas anteriormente para resolver la réplica de estructura en el gestor de base de datos *Oracle*. Se aportan elementos básicos para la solución, tales como la configuración, pasos o acciones a ejecutar para resolver las réplicas de estructuras, manuales o automáticos.

Tabla 1 Características de los replicadores.

Especificaciones/ Herramientas	DBMoto Cloud Edition	SymmetricDS	Oracle Streams	Replicador de datos Reko
Dirección de trasmisión de los datos				
Maestro - esclavo				
Multi - maestro	X	X	X	X
Gestores soportados				
PostgresSQL	X	X		X
MySQL	X	X		X
Oracle	X	X	X	X

Microsoft SQL Server	X	X	X	X
Otros gestores	X	X	X	
Técnica empleada para capturar cambios de esquemas				
Triggers		X		
Logs	X			
Script				
Mensajes			X	
Funciones				X

Fuente: elaboración propia.

1.3 Metodología de desarrollo

La metodología de desarrollo de *software* surge ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto (*software*). Logran la construcción de un sistema informático eficiente, que cumpla con los requerimientos planteados, 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".(24)

Para el desarrollo de la presente investigación se emplea la metodología de desarrollo AUP-UCI, debido a que es la metodología que ha designado la universidad para los proyectos de desarrollo, y está definida por el proyecto para el cual se desarrolla el proceso, además genera los artefactos que permiten obtener la documentación necesaria para una mejor comprensión de la herramienta a desarrollar.

Metodología AUP-UCI

Al no existir una metodología de *software* universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigiéndose así que el proceso sea configurable, se hace una variación de la metodología *AUP*, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI.(25)

De las cuatro fases que propone *AUP* (Inicio, Elaboración, Construcción, Transición) se decide para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando

el objetivo de la misma, se unifican las restantes tres fases de *AUP* en una sola, la cual se denomina Ejecución y se agrega la fase de Cierre.

Fases de AUP-UCI:

- **Inicio**: durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planificación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo, 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 mismo.(25)

Escenarios de AUP-UCI:

- **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).(25)

El proyecto Replicador de datos Reko emplea el escenario No 4 para modelar el sistema, el cual permite encapsular los requisitos funcionales en HU. Por tanto, se hará uso de esta técnica para encapsular los requisitos funcionales.

Roles de AUP-UCI, que reúnen el conjunto esencial de habilidades para realizar el desarrollo:

Tabla 2. Roles y artefactos.

Roles	Artefactos de salida
Jefe de proyecto	Plan de desarrollo de *software*. Método de estimación.
Planificador	Estándar de codificación.
Analista arquitecto de información (Opcional**)**	Historias de usuario. Modelo conceptual. Especificación de requisitos de *software*.
Desarrollador	Código fuente.
Administrador de la configuración	Lista de verificación de la configuración. Solicitud de cambio.
Stakeholder	Método de estimación. Solicitud de oferta de productos y de servicios.
Administrador de calidad	Diseño de casos de prueba. Plan de pruebas. Plan de desarrollo de *software*.
Probador	Diseño de casos de prueba.
Arquitecto de software	Modelo de diseño. Arquitectura de *software*.
Administrador de BD	

Fuente: Serie Científica de la Universidad de las Ciencias Informáticas.(26)

1.4 Herramientas y tecnologías

En este epígrafe se da tratamiento a los conceptos básicos relacionados a las herramientas y tecnologías que emplea el Replicador de datos Reko. Debido a que estas herramientas y tecnologías disminuyen la curvatura de aprendizaje y el tiempo de desarrollo, utilizándose desde versiones anteriores. No obstante, se tiene en cuenta que puede existir problemas de compatibilidad una vez que se lleve a cabo la integración del proceso para la réplica de estructuras al utilizar otras herramientas.

Lenguaje Unificado de Modelado 2.0

Lenguaje Unificado de Modelado (*UML*[30], por sus siglas en inglés) es un lenguaje estándar en el análisis y diseño de sistema de cómputo que permiten visualizar, especificar, construir y documentar los artefactos de un sistema con gran cantidad de *software*. *UML* proporciona una forma estándar de escribir los planos de un sistema, cubriendo tanto las formas conceptuales, tales como procesos del negocio y funciones del sistema, como las formas

[30] *Unified Modeling Language*

concretas, tales como las clases escritas en un lenguaje de programación específico, esquemas de *BD* y componentes *software* reutilizables. Está pensado para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código, así como generadores de informes. Además, brinda apoyo a la mayoría de los procesos de desarrollo orientados a objetos.(27)

***Visual Paradigm* para *UML* 8.0**

Es una herramienta CASE que aporta el modelado mediante *UML* y proporciona asistencia a los analistas, ingenieros de *software* y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un *software*. Permite visualizar todos los diagramas de clases, código inverso y generar documentación. Es una herramienta multiplataforma y permite la integración con *Eclipse IDE*[31].(28)

Java

Lenguaje de programación orientado a objeto (*OOP*, por sus siglas en inglés), robusto, de código abierto e independiente de la plataforma. Es capaz de gestionar la memoria automáticamente, 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.(29)

***Java Empresarial* 1.7 (*JEE*[32])**

Es un entorno independiente de la plataforma centrado en *Java* para desarrollar, crear e implementar en línea aplicaciones empresariales basadas en *web*. Consta de un conjunto de servicios, *APIs*[33] y protocolos que proporcionan la funcionalidad necesaria para desarrollar aplicaciones basadas en *web* de varios niveles. Proporciona un conjunto de especificaciones que aseguran la portabilidad de las aplicaciones.(30)

Eclipse KEPLER

Es una plataforma de desarrollo de código abierto basada en *Java*, tiene un conjunto de complementos, incluidas las Herramientas de Desarrollo de *Java* (*JDT*, por sus siglas en inglés). Su uso no se limita al lenguaje *Java*, incluye soporte para los lenguajes de programación como C/C++ y *COBOL*. (31)

[31] Entorno de Desarrollo Integrado, del inglés *Integrated Development Environment*. Es un programa compuesto por un conjunto de herramientas de programación que permite escribir código fuente, compilarlo y ejecutarlo sin necesidad de cambiar de aplicación.
[32] Java Empresarial, del inglés *Java Enterprse Edition.*
[33] Interfaz de programación de aplicaciones, del inglés *Application Programming Interface.*

***Apache ActiveMQ* 5.9.0**

Potente servidor de mensajería de código abierto escrito en *Java*. Fue diseñado para comunicarse a través de una serie de protocolos con el apoyo de diferentes lenguajes de programación, tales como: *Java*, *C++*, *C#*, entre otros.(32)

***Apache Tomcat* 7.0**

Apache Tomcat es un contenedor de *servlet*[34] usado en la implementación de referencia oficial para las tecnologías *Java Servlet* y *Java Server Pages* (*JSP*). Es desarrollado en un ambiente participativo y abierto.(33)

***Spring* 2.0**

Contenedor[35] y *framework* de peso ligero, basado en inyección de dependencias y orientación a aspecto, promueve el bajo acoplamiento pues los objetos reciben pasivamente sus dependencias en lugar de crearlas o buscar los objetos dependientes por sí mismos. Brinda facilidades para desarrollar una aplicación empresarial en *JEE*. Su aspecto modular lo hace flexible y configurable para cualquier tipo de aplicación.(34)

***Oracle* 11g**

Es básicamente una herramienta cliente/servidor para la gestión de *BD*, la gran potencia que tiene y su elevado precio hace que solo se vea en empresas muy grandes y multinacionales, por norma general. Ha sido diseñada para que las organizaciones puedan controlar y gestionar grandes volúmenes de contenidos no estructurados en un único repositorio con el objetivo de reducir los costes y los riesgos asociados a la pérdida de información. Incluye mejoras de rendimiento y de utilización de recursos, soporta aplicaciones de procesamiento de transacciones online (*OLTP*) y de data *warehousing* mayores y más exigentes. (35)

1.5 Consideraciones del capítulo

En consecuencia, al análisis elaborado en este capítulo 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 muestra la metodología y las diferentes herramientas que se utilizarán en el desarrollo del proceso, las cuales guiarán el proceso de desarrollo.

[34] Programa Java que se ejecuta dentro de un servidor Web. Reciben y atienden las solicitudes de los clientes web.
[35] Interfaz entre el componente y la plataforma sobre la que se ejecuta. Facilita los servicios que éste necesita para su funcionamiento.

- Se empleó la metodología de desarrollo AUP-UCI y herramientas utilizadas para el desarrollo del Replicador de datos Reko en versiones anteriores lo cual reducirá la curva de aprendizaje y el tiempo de desarrollo.

CAPÍTULO 2. PROPUESTA DE SOLUCIÓN

En este capítulo se dan a conocer los requisitos funcionales y no funcionales, las características de la solución propuesta para resolver el problema de la investigación, se realiza una descripción de la arquitectura propuesta, y los requerimientos del sistema. Además, mediante la metodología de desarrollo de *software* AUP-UCI, se 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.(22)

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.(22)

Cuando se trabaja con el gestor de *BD Oracle* y se aplican acciones de *DDL* (*create*, *alter*, *drop*) sobre la estructura en la *BD* se necesita reiniciar la aplicación, ya que dichas acciones no son reconocidas automáticamente en la configuración de réplica de datos, siendo el cambio invisible en la configuración de réplica de estructura. Como no actualiza inmediatamente los cambios realizados sobre la configuración de réplica, provoca que disminuya el aprovechamiento laboral del personal involucrado en este proceso debido que deben solucionar los problemas de forma manual. Posibilitando que se introduzcan errores en la estructura de la *BD*, ocasionando inconsistencias entre los datos que se replican, tablas o datos incompletos e incluso problemas de integridad referencial, todo esto afecta la coherencia de los datos.

Como solución a estas limitantes, se propone extender el proceso de réplica de estructura entre *BD* gestionadas en *Oracle* al Replicador de datos Reko.

2.2 Modelo de dominio

Un modelo del dominio es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés, muestra (a los modeladores) clases conceptuales

significativas en un dominio de problema; es el artefacto más importante que se crea durante el análisis orientado a objetos.(36)

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

- Objetos del dominio o clases conceptuales.
- Asociaciones entre las clases conceptuales.
- Atributos de las clases conceptuales.(36)

2.2.1 Diagramas de clases del dominio

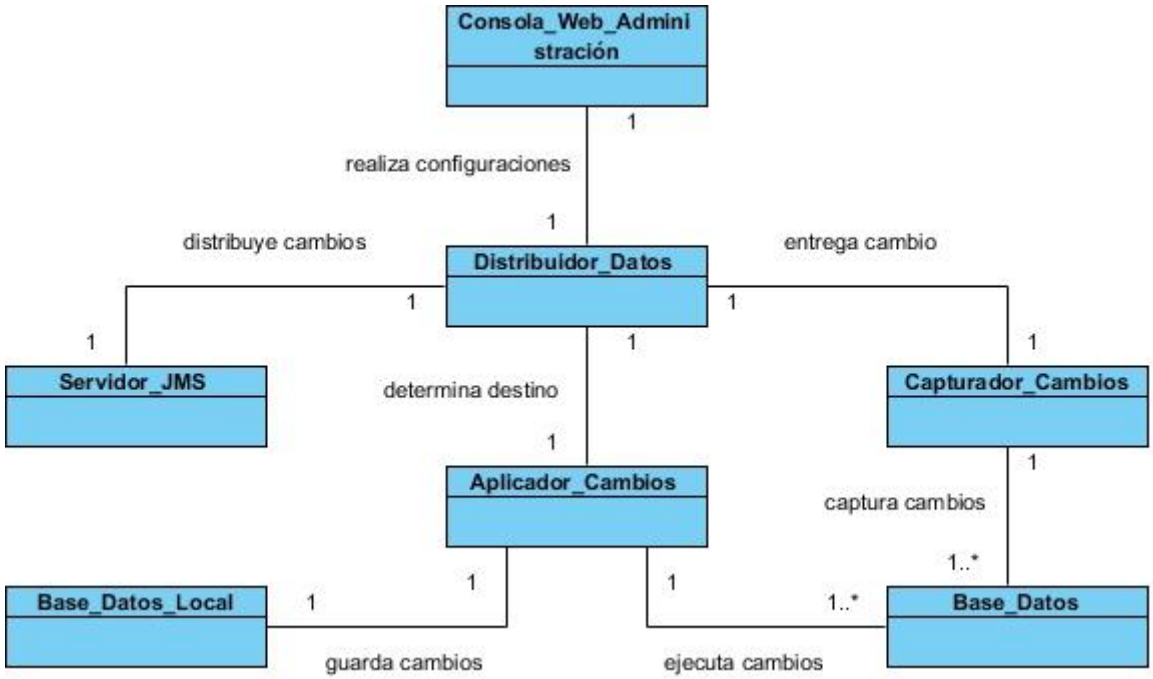


Figura 2 Modelo de dominio.
Fuente: elaboración propia.

2.2.2 Descripción de las clases del modelo de dominio

A continuación, se describen las clases del modelo de dominio para una mayor comprensión:

- **Consola_Web_Administración**: representa la interfaz del *software*. Permite realizar las configuraciones principales del *software* como el registro de nodos, configuración de las tablas y esquemas a replicar, configuración de conflictos, resolución de conflictos y el monitoreo del funcionamiento del *software*.
- **Distribuidor_Datos**: determina el destino de cada configuración de cambio, se responsabiliza de su llegada.
- **Capturador_Cambios**: se encarga de capturar los cambios realizados a la

Base_Datos y de entregarlos al Distribuidor_Datos.

- **Servidor_JMS:** representa el servidor de mensajería, utilizado como punto intermedio para la distribución de la información enviada.
- **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 desde otro nodo, donde se capturan y clasifican los conflictos generados.
- **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 para un sistema establecen con detalle las funciones, servicios y restricciones operativas del sistema. Estos requisitos reflejan la necesidad de los clientes de un sistema que ayuda a resolver problemas como el control de un dispositivo, hacer un pedido o encontrar información. El proceso de descubrir, analizar, documentar y verificar esos servicios y restricciones se denomina ingeniería de requisitos.(37)

2.3.1 Requisitos funcionales

Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que este debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos los requisitos funcionales también puedes declarar explícitamente lo que el sistema no debe hacer.(37)

Los requisitos funcionales identificados se representan a continuación:

- **RF1**: Crear estructuras de control.
- **RF2**: Capturar los cambios de estructura.
- **RF3**: Eliminar acciones de estructuras replicables.
- **RF4**: Aplicar cambios de estructura.

2.3.2 Requisitos no funcionales

Son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. Los requisitos no funcionales a menudo se aplican al sistema en su totalidad. Normalmente apenas se aplican a características o servicios individuales del sistema.(37)

Los requisitos no funcionales del proceso a desarrollar son equivalentes a los del *software* Replicador de datos Reko al cual se integrará. Estos se describen a continuación ver tabla 3:

Para una mayor comprensión de los requisitos no funcionales ver Anexos del 1 al 8.

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	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.
RnF4	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.
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.

2.4 Propuesta de solución

Para la solución del problema planteado se utilizarán las funcionalidades del componente "Consola Web de Administración" con el objetivo de iniciar la configuración de réplica de estructura, dando paso a la creación de las estructuras de control. Las cuales contendrán información referente a los cambios de estructura por acciones *DDL* (*create*, *alter* y *drop*) producida en la estructura de la *BD Oracle* y se crearan los *triggers* para capturar estas informaciones.

El componente "Capturador de cambios" registra los cambios realizados en una estructura de control (tabla espejo[36]), crea un objeto de tipo grupo de estructura replicable (*RSG*, *Replicable Structure Group*), el cual contendrá un grupo de objetos de la clase (*RSA*, *Replicable Structure Action*). El "Distribuidor de Datos" construye cada *RSA* a partir de la información almacenada en la tabla espejo y el **metadata** de la *BD* al cual accede a través de la clase "**MetadataManager**" que tiene implementado Reko. Cada *RSA* almacena la información conveniente al cambio realizado. 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*, establece los destinos hacia donde se envían cada *RSG*, responsabilizándose además de la propagación de estos empleando el servidor de *JMS* ActiveMQ.(22)

Una vez enviado cada *RSG* a su destino, se deberá eliminar las acciones de estructura replicable contenidas en la tabla espejo. En cada nodo destino, una vez recibido un *RSG*, pasará al "Aplicador de Cambios" el cual ejecutará las acciones contenidas en los *RSG* en la *BD* destino gestionadas con *Oracle*.

[36] 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, así como el usuario que ejecuta la acción.

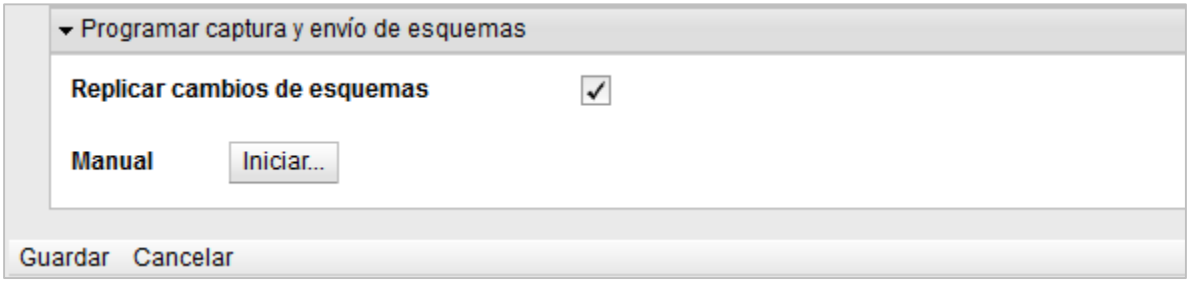
2.5 Historias de usuario (HU)

Las HU constituyen una manera sencilla de encapsular los requisitos del sistema, están diseñadas para un fácil entendimiento minimizando los detalles, de forma que se pueda determinar el costo de la implementación, de acuerdo con las necesidades del sistema y son utilizadas por las metodologías ágiles.

2.5.1 Descripción de las HU

A continuación, se muestran ejemplos de HU definidas en la solución, (ver tabla 4 y 5).
Las demás representaciones de las HU se hallan en los Anexos del 9 al 10.

Tabla 4 HU1 Crear estructuras de control.

Historias de usuario	
Número: HU 1	**Nombre del requisito**: Crear estructuras de control
Programador: Norberto Williams Barrientos Trenal	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 29 días
Riesgo en Desarrollo: Plan de riesgos	**Tiempo Real**: 174 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 también los *triggers* que capturaran estas informaciones. La estructura de control **"reko_structure_control_table**"" (tabla espejo) 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.

Tabla 5 Capturar los cambios de estructura.

Historias de usuario	
Número: HU 2	**Nombre del requisito**: Capturar los cambios de estructura.
Programador: Norberto Williams Barrientos Trenal	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 29 días
Riesgo en Desarrollo: Plan de riesgos	**Tiempo Real**: 174 horas
Descripción: Para capturar los cambios de estructura se crean tres triggers, el trigger "**trigger_capture_create**" el cual captura los cambios de estructura por la acción *DDL* (*create*), el trigger "**trigger_capture_drop**" captura los cambios de estructura por el comando *DDL* (*drop*) y los *DDL* (alter) serán capturados por el trigger "**trigger_capture_alter**".	
Observaciones: La trigger se crearán 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.

2.6 Arquitectura del Replicador de datos REKO

La arquitectura de *software* de un programa o sistema de cómputo es la estructura o las estructuras del sistema, que incluyen los componentes del *software*, las propiedades visibles externamente de esos componentes y las relaciones entre ellos.(38)

Un patrón arquitectónico, impone una regla sobre la arquitectura, pues describe la manera en que el software manejara algún aspecto de su funcionalidad al nivel de la infraestructura. Tienden a abarcar aspectos específicos del comportamiento dentro del contexto de la arquitectura. Se usan junto con un estilo arquitectónico para determinar la forma de la estructura general de un sistema.(38)

2.6.1 Estilo y patrones arquitectónicos

El estilo arquitectónico **Llamada** y **retorno** es el empleado en la solución propuesta ya que se obtiene una estructura del programa que resulta relativamente fácil de modificar y cambiar de tamaño. Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.(39)

Sus características son:

- Hilo de control simple soportado por los lenguajes de programación.
- Usa una estructura implícita de subsistemas.
- Razonamiento jerárquico, cambios en una subrutina implican cambios en las subrutinas que hacen la invocación.
- Pretenden incrementar el desempeño distribuyendo el trabajo en múltiples procesadores.(39)

Arquitectura basada en componentes

Un componente es una unidad de composición de aplicaciones *software*, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio.(40)

Una arquitectura basada en componentes describe una aproximación de ingeniería de *software* al diseño y desarrollo de un sistema. Esta arquitectura se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas. Esto provee un nivel de abstracción mayor que los principios de orientación por objetos y no se enfoca en asuntos específicos de los objetos como los protocolos de comunicación y la forma como se comparte el estado.(41)

Las definiciones antes mostradas se aplican 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 reemplazados 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 "**trigger**", correspondiente al componente Capturador de cambios, se le realizarán modificaciones, pues hasta el momento su funcionamiento se basa en realizar configuraciones de réplica y conflictos de estructuras, este proceso únicamente se realiza para el gestor *PostgreSQL*, ahora se extenderá su funcionamiento para resolver configuraciones de réplica para el Gestor de *BD Oracle*. Con este mismo fin serán modificados los paquetes del Distribuidor, Aplicador y Administrador.

Arquitectura basada en capas

La arquitectura basada en capas se enfoca en la distribución de roles y responsabilidades de forma jerárquica proveyendo una forma muy efectiva de separación de responsabilidades. El

rol indica el modo y tipo de interacción con otras capas, y la responsabilidad indica la funcionalidad que está siendo desarrollada.(39)

El componente Administración responde a un modelo multicapas, donde cada capa se define como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Además de estar separadas lógica y estructuralmente, las capas se encuentran separadas de manera física.(42)

- 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.
- La **capa de negocio** es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina además como lógica del negocio porque es aquí donde se establecen todas las reglas que deben cumplirse.(42)

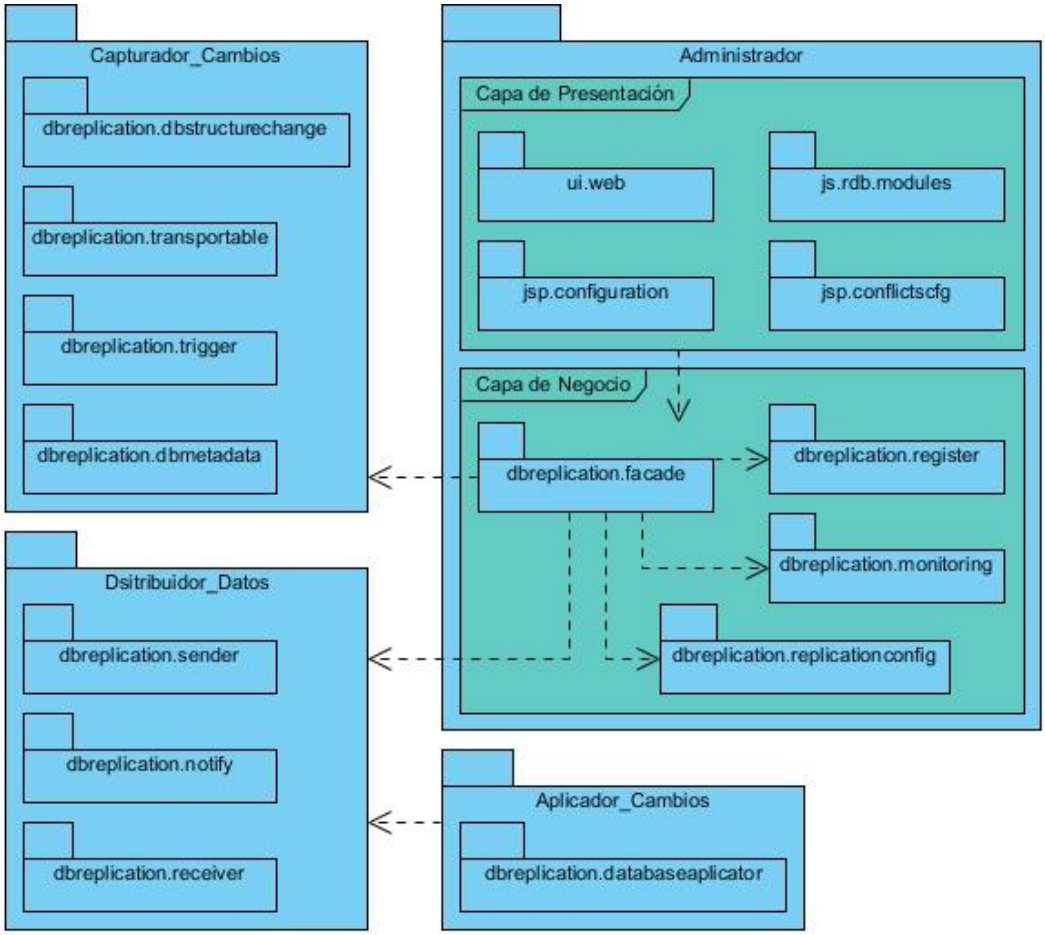


Figura 3 Principales componentes del proceso de réplica de estructura en el Replicador de datos Reko v4.0. Fuente: Componente para el envío y recepción de datos del proceso de réplica de estructura para el software de replicación Reko.(22)

2.6.2 Patrones de diseño

Un patrón de diseño es una solución repetible a problemas típicos y recurrentes en el diseño del *software*. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan. No son un diseño terminado que puede traducirse directamente a código, sino más bien una descripción sobre cómo resolver el problema, la cual puede ser utilizada en diversas situaciones.(43)

Eric Gamma[37] define los patrones de diseño como:
"Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí para resolver un problema de diseño general en un contexto particular".(43)

Se emplearon los patrones de asignación de responsabilidades para el desarrollo del proceso, conocidos como patrones ***GRASP***[38] los cuales describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.(44)

- **Experto**: en el desarrollo del proceso de réplica de estructura entre *BD* gestionadas en *Oracle* para el Replicador de datos Reko se utilizó el patrón experto, dado que cada clase cuenta con la información necesaria para cumplir con la responsabilidad que se le establece. Ejemplo, la clase "**DBStructureChangeManager**" cuenta con la información necesaria para cumplir la responsabilidad de capturar los cambios de estructura y la creación de los *RSG*.
- **Controlador**: un controlador representa un receptor o manejador de todos los eventos del sistema. Define además el método de su operación. En el proceso desarrollado se utilizan distintas clases controladoras como la "**AplicatorManager**" y la clase "**DBStructureChangeManager**".
- **Bajo acoplamiento**: las clases se comunican solamente con las clases necesarias para desarrollar cada flujo de evento.
- **Alta cohesión**: como cada clase tiene un conjunto de funcionalidades relacionadas directamente con la entidad que representan, no realizan un trabajo enorme y por tanto pueden ser calificadas como de alta cohesión.
- **Polimorfismo**: la responsabilidad de definir la variación de comportamiento basado en tipos se asigna a aquellos para los cuales esta variación ocurre. Esto se logra mediante el uso de operaciones polimórficas. Este patrón se evidencia en la clase

[37] Informático suizo, empleado de Microsoft, fue el creador de *JUnit*, junto a Kent Beck (máster en ciencias en el área de ciencias de la computación de la Universidad de Oregon). Formó parte del *grupo de los cuatro* (*Gang of Four* o, simplemente, *GoF*) que popularizaron los patrones de diseño.
[38] Patrones Generales de *Software* para Asignar Responsabilidades, del inglés *General Responsibility Assignment Software Patterns.*

"**HibernateDialect**" la cual es una interfaz para métodos implementados por dialectos (**Oracle9iDialect**) de *BD* con el propósito de manejar tipos de *BD* y *SQL* estándar y no estándar.(44)

También se emplearon los patrones ***GOF***[39] los cuales se clasifican en dependencia del propósito para los que hayan sido definidos: creación, estructurales y de comportamiento.(45)

- **Fachada**: proporciona una interfaz unificada simple, que permite acceder a una interfaz o grupo de interfaces de un subsistema. Este patrón se utiliza para reducir la dependencia entre clases y ofrece un punto de acceso, de manera que, si estas son modificadas o se sustituyen por otras, solamente se tiene que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones cliente. Por ejemplo, Reko tiene el componente *facade* que aprovisiona estas ventajas, la clase "**TransactionProcessor**" del paquete "**facade**", ofrece un punto de acceso para iniciar el envío de los *RSG* y el inicio de la aplicación de los cambios en las *BD*.(45)

Se utilizó además el patrón ***DAO***[40] para la solución propuesta, en el cual estarán divididas las responsabilidades en la aplicación en dos aspectos, clases que se encargaran de la responsabilidad de persistencia y en otro las clases que se encargaran de la lógica del negocio. No debe haber ningún tipo de código que acceda 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*.(46)

2.7 Modelo de diseño

Un modelo de diseño es una abstracción del sistema, especificando el sistema modelado desde un cierto punto de vista y en un determinado nivel de abstracción. Describe la realización de las HU, los subsistemas y las clases del sistema y cómo interactúan.(47)

2.7.1 Diagramas de paquetes

Un diagrama de paquetes muestra cómo un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema.(48)

[39] Banda de los Cuatro, del inglés *Gang-Of-Four*.
[40] Objeto de Acceso a Datos, del inglés *Data Access Object*.

Se muestra el diagrama de paquetes del proceso propuesto.

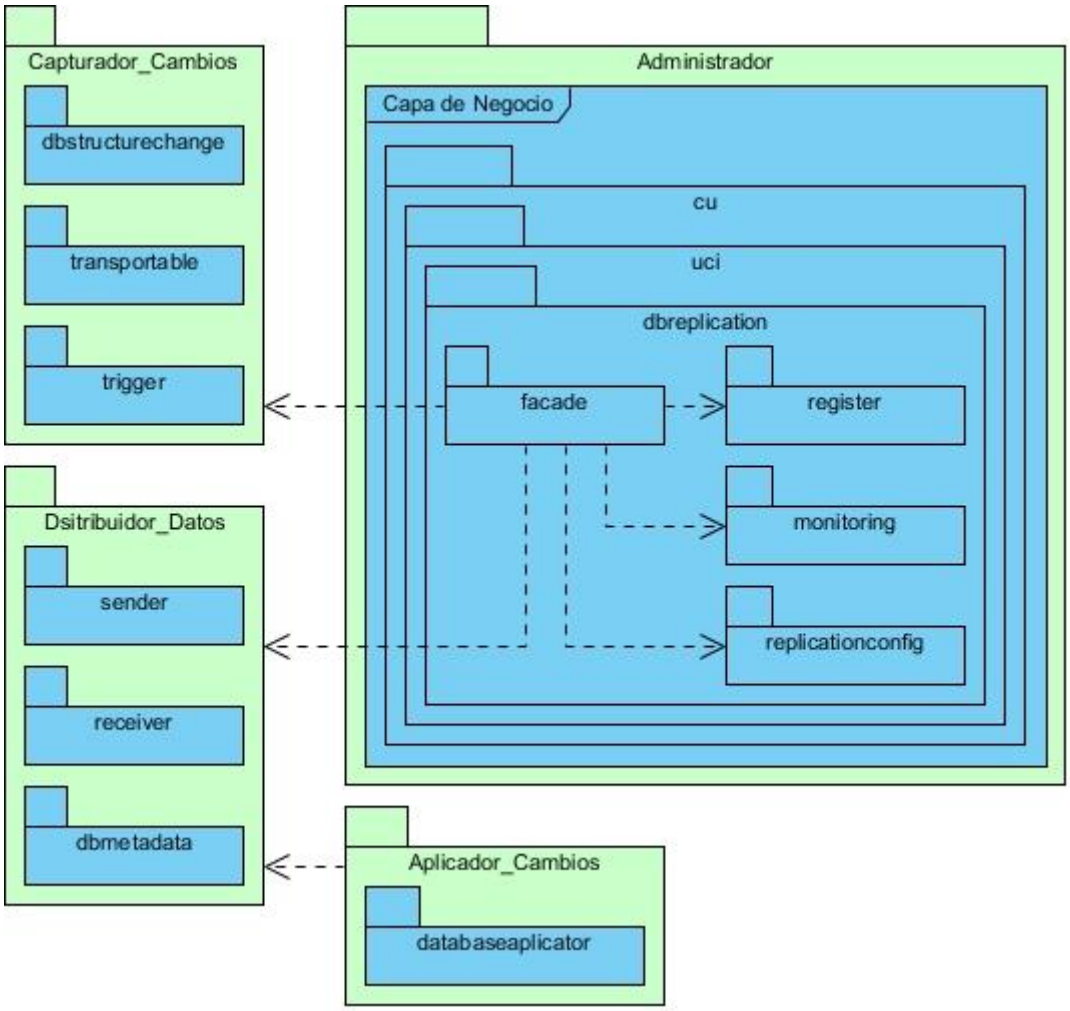


Figura 4 Diagrama de paquetes de los RF.
Fuente: elaboración propia.

2.7.2 Diagramas de clases de diseño

Los diagramas de clases exponen un conjunto de interfaces, colaboraciones y sus restricciones. Se utilizan para modelar la vista de diseño estática de un sistema. Son importantes para visualizar, especificar, documentar modelos estructurales y construir sistemas ejecutables con la aplicación de ingeniería directa e inversa.(49)

A continuación, se muestran los diagramas de clases diseñados para los requisitos funcionales del sistema, (ver figura 5 y 6).

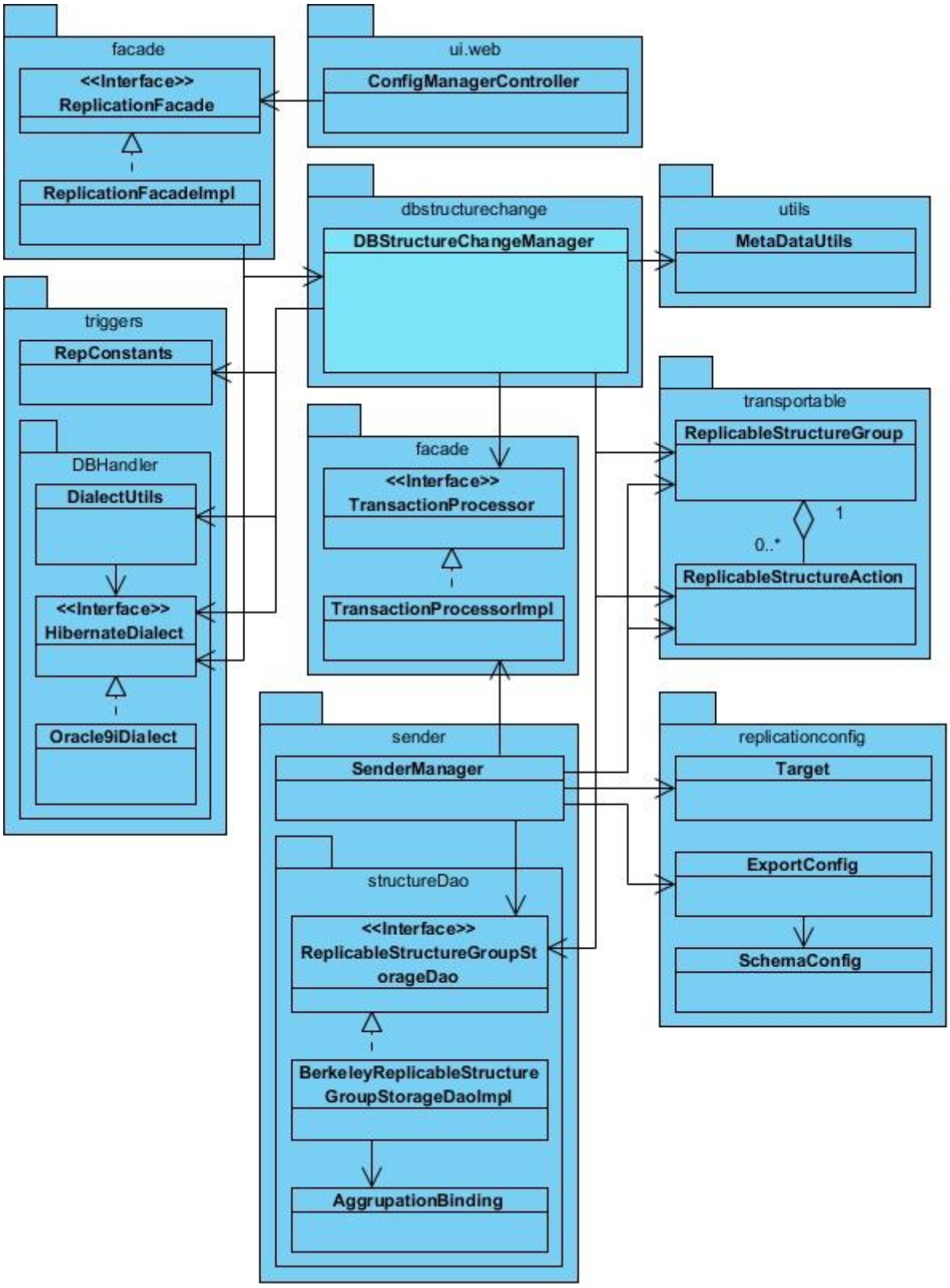


Figura 5 Diagrama de Clases para los RF1, RF2 y RF3.
Fuente: elaboración propia.

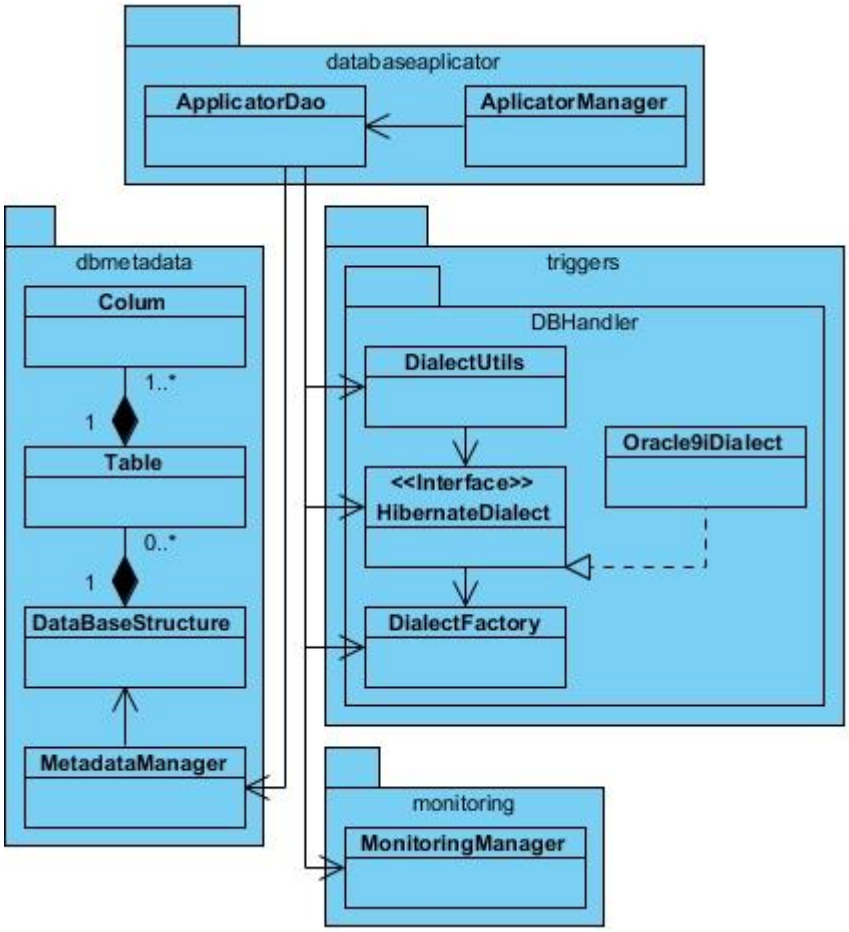


Figura 6 Diagrama de Clases para el RF4.
Fuente: Componente para el envío y recepción de datos del proceso de réplica de estructura para el software de replicación Reko.(22)

2.7.3 Descripción de las clases del diseño

Tabla 6 Descripción textual de la clase ReplicableStructureGroup

Descripción de la clase Oracle9iDialect	
Nombre: `Oracle9iDialect`	
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 usuario o esquema.
Nombre:	`renameTable(oldTableName: String, newTableName: String): String`

Descripción:	Obtiene la sentencia sql de renombrar 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:	`alterTypeColum(tableName: String, columName: String, newType: String): String`
Descripción:	Obtiene la sentencia sql de cambiarle el tipo de dato a 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:	`alterDropConstraint(tableName: String, constraintName: String): String`
Descripción:	Obtiene la sentencia sql para eliminar un constraint.
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:	`getCantActionstoReplics(): String`

Descripción:	Obtiene la sentencia sql para saber la cantidad de acciones que tiene la tabla espejo.
Nombre:	`getOfMirrorTable(cantElementsToRead: int): String`
Descripción:	Obtiene la sentencia sql para conseguir de la tabla espejo una cantidad de acciones igual a la especificada por parámetro.
Nombre:	`deleteTupleOfMirrorTable(id: int): String`
Descripción:	Obtiene la sentencia sql para eliminar de la tabla de control la acción especificada por parámetro.
Nombre:	`columnDefinition(colum: cu.uci.dbreplication.dbmetadata.Colum): String`
Descripción:	Obtiene el nombre de la columna y el tipo de dato.
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:	`String getColumTypeName(Type: String): String`
Descripción:	Obtiene el tipo de dato de la 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:	`getFkTable(fkConstraintName: String): String`
Descripción:	Obtiene la sentencia sql para saber el nombre de la tabla que ha sido referenciada dado el nombre de su constraint FK.
Nombre:	`createStructureControlTableSQL():String`
Descripción:	Obtiene la sentencia sql para crear la tabla de control para las estructuras.
Nombre:	`crateSequence(nameSequence: String): String`
Descripción:	Obtiene la sentencia sql para crear una secuencia.
Nombre:	`createSchemaRole(nameSchemaRole: String): String`

Descripción:	Obtiene la sentencia sql para darle roles a un usuario o esquema.
Nombre:	`getPkName(pkConstraintName: String): String`
Descripción:	Obtiene la sentencia sql para saber el nombre de la columna que es llave primaria PK.

Fuente: elaboración propia

Las restantes descripciones de clases se pueden ver en los Anexos del 9 al 13, se recurrió a métodos ya implementados y en varios casos se implementaron nuevas operaciones.

2.8 Modelo de despliegue

Muestra la configuración de los nodos de procesamiento en tiempo de ejecución, los links de comunicación entre ellos, y las instancias de los componentes y objetos que residen en ellos. El modelo de despliegue se utiliza para capturar los elementos de configuración del procesamiento y las conexiones entre esos elementos. También se utiliza para visualizar la distribución de los componentes de *software* en los nodos físicos.(50)

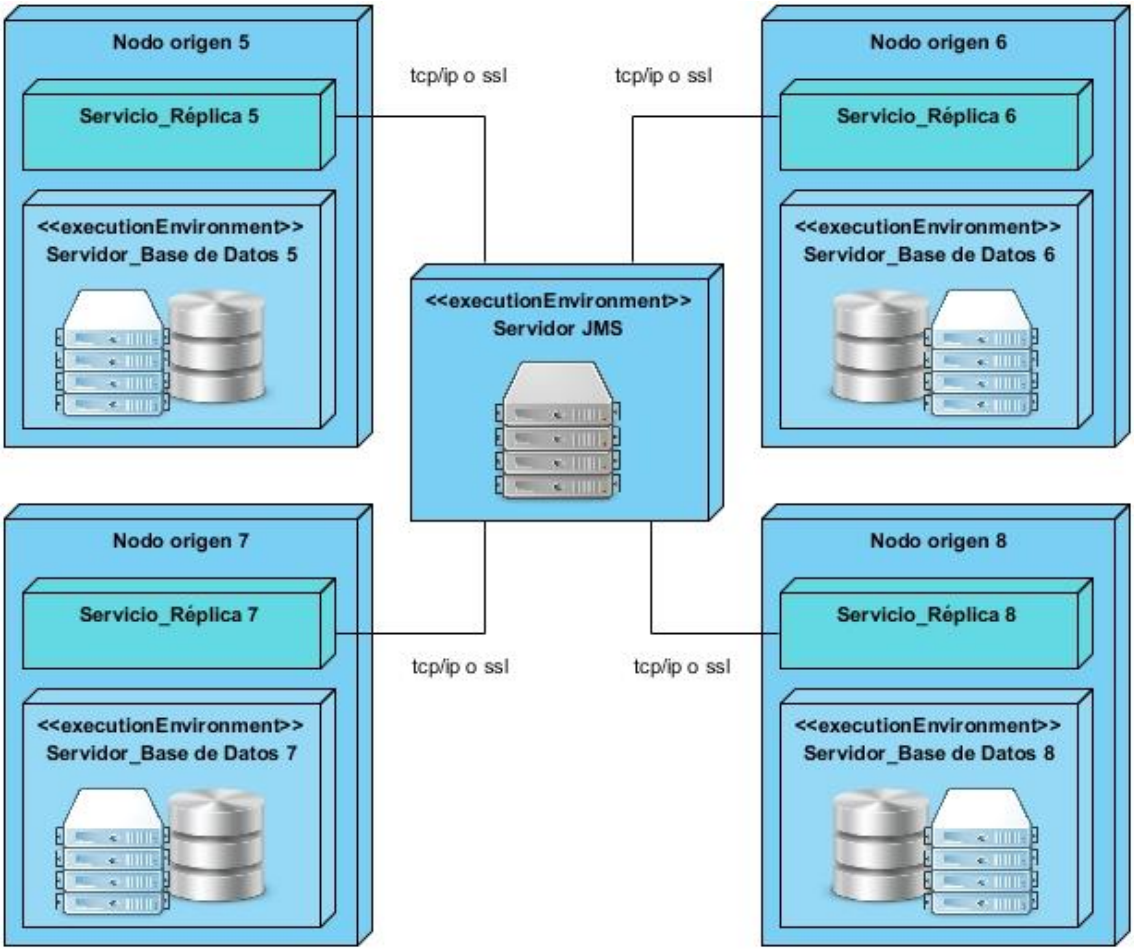


Figura 7 Diagrama de despliegue.
Fuente: elaboración propia.

2.9 Consideraciones del capítulo

Una vez realizado el análisis y el 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 acciones por la ejecución de consultas *DML* y *DDL* en este caso esta sólo para *PostgreSQL*, ahora se extiende su funcionamiento para permitir la actualización automática de los cambios producidos en la *BD* por la ejecución de consultas *DDL* para *Oracle*.
- El proceso a implementar estará compuesto por tres partes fundamentales: una encargada de la captura de los cambios de estructuras, una de eliminación de las acciones y otra de la aplicación de los cambios de estructuras en las *BD Oracle* destino.

CAPITULO 3. IMPLEMENTACIÓN Y VALIDACIÓN

Una vez diseñada la propuesta de solución, se procede a la implementación de cada una de las funcionalidades, así como la representación de los diagramas de componentes. Muestra el código fuente de una de las principales clases y las pruebas aplicadas al sistema para demostrar la validez del mismo.

3.1 Modelo de implementación

El modelo de implementación describe cómo los elementos del modelo de diseño, se implementan en términos de componentes, ficheros de código fuente, ejecutables y binario, bibliotecas dinámicas, páginas *web*, interfaces y un conjunto de relaciones de dependencia, generalización, asociación y realización.(51)

3.1.1 Diagramas de componentes

Los diagramas de componentes muestran las organizaciones y dependencias lógicas entre componentes de *software*, sean estos de código fuente, binarios, archivos, bibliotecas cargadas dinámicamente o ejecutables. Modelan la vista estática de un sistema.(51)

Como Reko presenta una arquitectura basada en componentes, se emplearon los subsistemas de implementación para realizar los diagramas de implementación.

Para entender la composición física de la implementación del proceso, se exponen algunos diagramas de componentes agrupados por los subsistemas implicados en la solución (ver figuras 8, 9 y 10). El restante diagrama de componente relacionado con la solución se puede ver en el anexo 14.

Los subsistemas de implementación que se muestran a continuación se relacionan a través de la interfaz “**TransactionProcessor**” (ver Anexo 15).

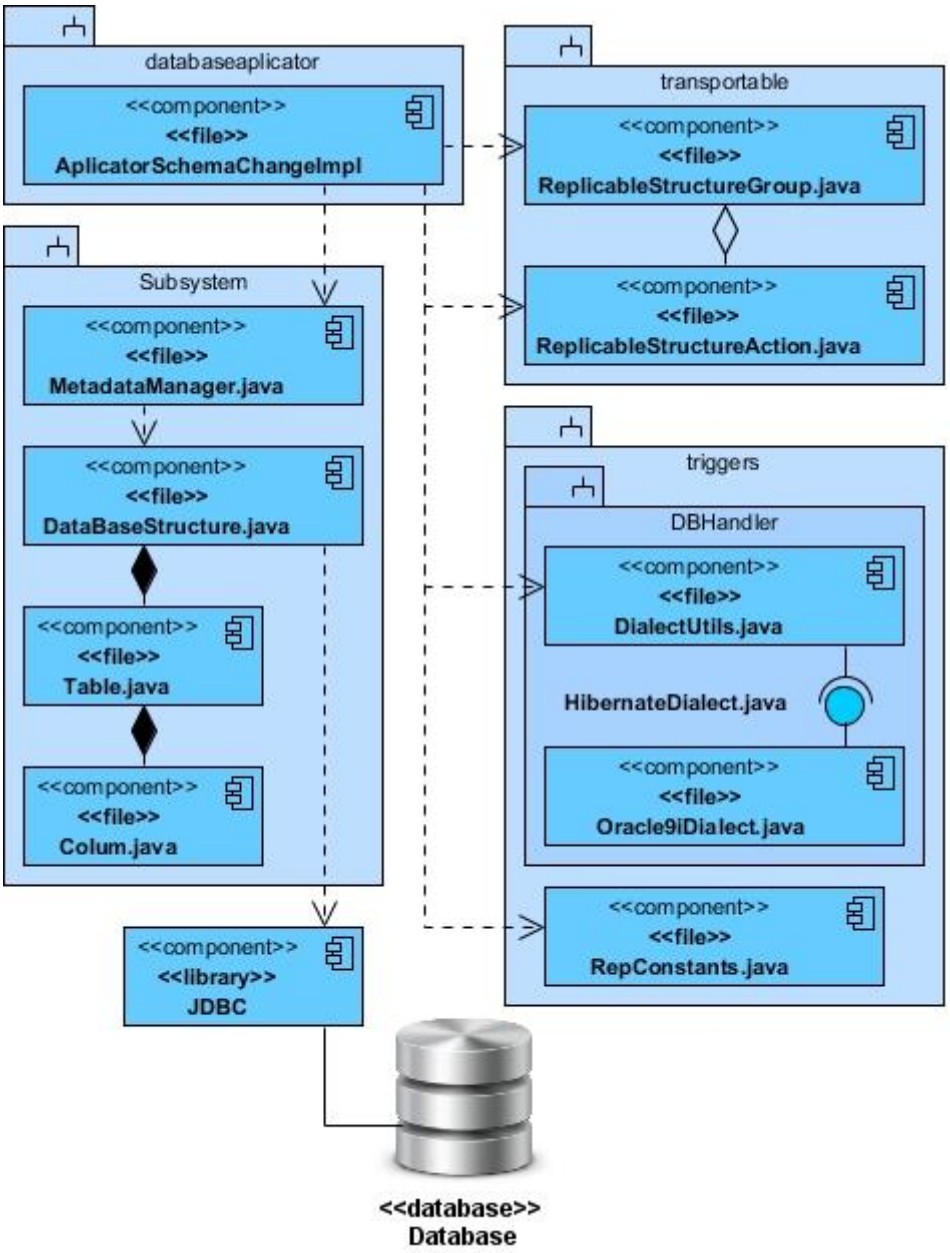


Figura 8 Diagrama de componentes para el subsistema Capturador de cambios del Replicador de datos Reko v4.0. Fuente: Componente para el envío y recepción de datos del proceso de réplica de estructura para el software de replicación Reko.(22)

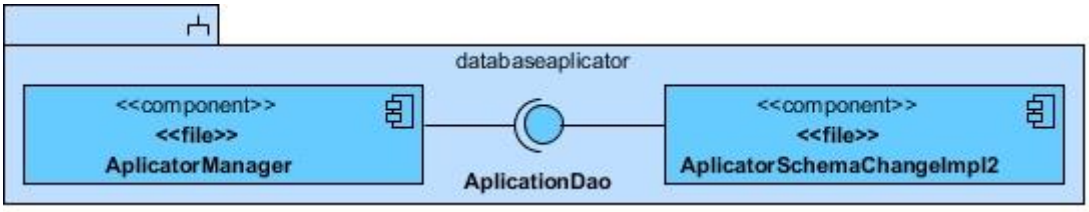


Figura 9 Diagrama de componentes para el subsistema Aplicador de cambios del Replicador de datos Reko v4.0. Fuente: Componente para el envío y recepción de datos del proceso de réplica de estructura para el software de replicación Reko.(22)

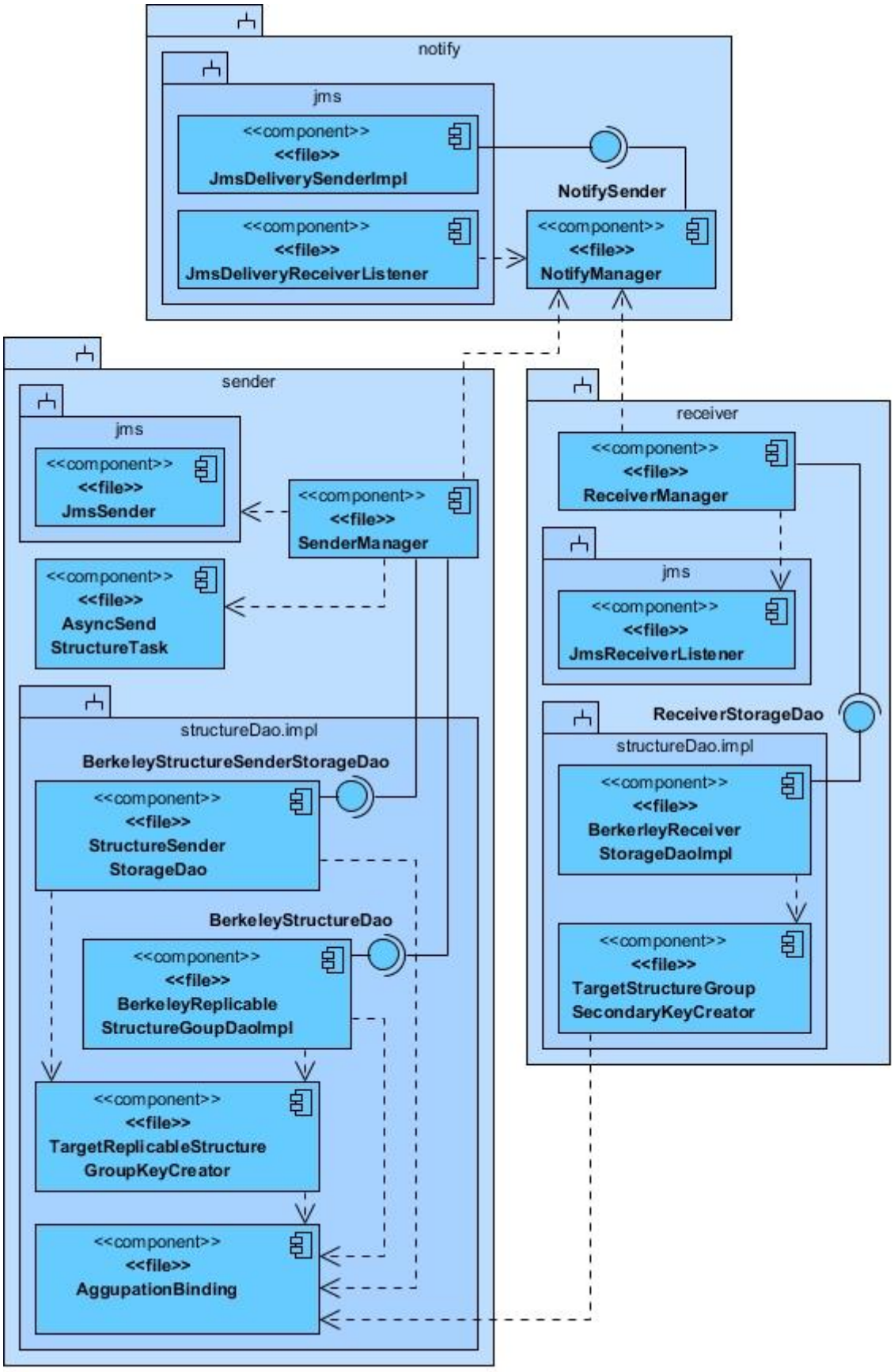


Figura 10 Diagrama de componentes subsistema Distribuidor de cambios del Replicador de datos Reko v4.0. Fuente: Componente para el envío y recepción de datos del proceso de réplica de estructura para el software de replicación Reko.(22)

3.2 Código fuente

El código fuente se define como el conjunto de líneas de textos, que son las directrices 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 fácilmente. Para esta traducción se emplean los llamados compiladores, ensambladores, intérpretes, entre otros.(52)

Estándar de codificación *java*

Las técnicas de codificación incorporan muchos aspectos del desarrollo del *software*. Aunque generalmente no afectan a la funcionalidad de la aplicación, sí contribuyen a una mejor compresión del código fuente. En esta fase se tienen en cuenta todos los tipos de código fuente, incluidos los lenguajes de programación, de secuencias de comandos, de marcado o de consulta.(53)

Los estándares de codificación son importantes para los programadores por un gran número de razones:

- El 80% del coste del código de un programa va a su mantenimiento.
- Casi ningún *software* lo mantiene toda su vida el auto original.
- Los estándares de código mejoran la lectura del *software*, permitiendo entender código nuevo mucho más rápidamente y más a fondo.
- Si distribuyes tu código fuente como un producto, necesitas asegurarte de que está bien hecho y presentado como cualquier otro producto.(54)

A continuación, se expone un ejemplo de un estándar de codificación *java* para el desarrollo del proceso de réplica de estructuras entre *BD* gestionadas en *Oracle* para el Replicador de datos Reko, los restantes estándares de codificación están reflejados en el Anexo 25.

Estándares de indicadores:

Variables y métodos: Para estos identificadores haremos uso de la variante (*lowerCamelCase*)**.** Empiezan con minúsculas y si estos identificadores están compuestos por varias palabras las siguientes empezarán con mayúscula.

Variables:

- `String tableName;`

Métodos:

Los nombres de método deben iniciar con un verbo, ejemplo:

- `public void addSentence(…) {}`

Los obtenedores de campos privados en las clases tienen el prefijo "**get**", ejemplo:

- `public String getCleanTableName(…) {}`

Los modificadores de campos privados en las clases tienen el prefijo "**set**", ejemplo:

- `public void setSqlMetadataManager(…) {}`

Los obtenedores con el resultado de booleano tienen el prefijo "**is**", ejemplo:

- `protected boolean isReadyToCapture(…) {}`

Se muestra a continuación el siguiente fragmento de código corresponde a la clase "**DBStructureChangeManager**":

- El método "**buildingReplicableStructureGroup(Connection connection)**" construye los grupos de estructuras que serán replicados, el mismo se ha modificado de tal forma que se logra obtener el id (**ID_STRUCTURE_CONTROL_TABLE**) de la RSA que será eliminada de la tabla de control, cuando se utilice el dialecto "**Oracle9iDialect**".

Tabla 7 Código fuente del método que construye los RSG

DBStructureChangeManager

```
public void buildingReplicableStructureGroup() throws SQLException {
  try {
      HibernateDialect dialect = null;
      dialect = DialectFactory.getDialect(sqlMetadataManager);
      int cantActions = 0;
      String cantActionstoReplics = DialectUtils
         .getCantActionstoReplics(dialect);
      Statement statement = connection.createStatement();
      ResultSet resultCount = statement
         .executeQuery(cantActionstoReplics);
      if (resultCount.next()) {
        cantActions = resultCount.getInt(1);
      }
      resultCount.close();
      while (cantActions > 0) {
         // construir grupos teniendo en cuentas la cantidad
         // máxima de acciones q puede contener un grupo
        int cantElementsToRead = Integer.parseInt(GeneralConfig
              .getCANT_ACTIONS_IN_GROUP());
         // consulta para acceder a la tabla espejo
         // se obtienen los primeros cantElementsToRead
         String valuesOfMirrorTable = DialectUtils.getOfMirrorTable
```

```
        (dialect, cantElementsToRead);
        ResultSet result = statement
             .executeQuery(valuesOfMirroTable);
        // Crear RSG con id, tipo de replicación y el nodo
        // que lo envía
        ReplicableStructureGroup replicableStructureGroup =
             new ReplicableStructureGroup(
             getNewReplicableStructureGrupId(),
             ReplicableStructureGroup.REPLICATOR_WITH_FUNCTION,
             GeneralConfig.getNodeId());
        // se recorren las primeras #cantElementsToRead
        // acciones 1x1 while para recorrer las tuplas
        // result.next para recorrer las columnas de c/tupla
        while (result.next()) {
        // construir las acciones q se add a los RSG
           ReplicableStructureAction replicableStructureAction =
                getReplicaleStructureActionByResultSet(result);
           replicableStructureGroup
               .addSentence(replicableStructureAction);
           int id;
           // Obtiene el id de la tabla de control de Oracle
           if (dialect instanceof Oracle9iDialect) {
               id = result.getInt(RepConstants
                    .ID_STRUCTURE_CONTROL_TABLE);
           }
           else {
               id = result.getInt(RepConstants.REPLIC_ID);
           }
           Statement statementDelete = connection
                 .createStatement();
           String deleteRepAction = DialectUtils
                 .deleteTupleOfMirrorTable(dialect, id);
           MetaDataUtils.runDDL(statementDelete, deleteRepAction);
           statementDelete.close();
        }
        result.close();
        // persistir y enviar
        replicableStructureGroupStorageDao
            .persist(replicableStructureGroup);
        TransactionProcessor.sendReplicableStructureGroup(
             replicableStructureGroup);
        // Si se le insertaron nuevas acciones
        // Mientras se enviaban las anteriores
        ResultSet newCount = statement
             .executeQuery(cantActionstoReplics);
        if (newCount.next())
            cantActions = newCount.getInt(1);
        newCount.close();
      }
      statement.close();
```

```
        connection.rollback();
        connection.close();
        logger.info("Finalizado proceso de construcción de la
        agrupación ");
    } catch (Exception e) {
    // TODO Auto-generated catch block
    logger.error("Error con la DB");
    connection.rollback();
    connection.close();
    e.printStackTrace();
  }
}
```

Fuente: elaboración propia

3.3 Evaluación del proceso usando métricas de *software*

Las métricas de *software* son una medida cuantitativa que permite a los desarrolladores tener una visión profunda de la eficacia del proceso del *software* y de los proyectos que dirigen utilizando el proceso como un marco de trabajo. Se reúnen los datos básicos de calidad y productividad. Estos datos son entonces analizados, comparados con promedios anteriores, y evaluados para determinar las mejoras en la calidad y productividad. Las métricas son también utilizadas para señalar áreas con problemas de manera que se puedan desarrollar los remedios y mejorar el proceso del *software*.(55)

Las métricas empleadas están diseñadas para evaluar los siguientes atributos de calidad:

- **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.(55)

Se emplean las métricas “Tamaño Operacional de la Clase” y “Relaciones entre Clases” como herramientas para evaluar la calidad del diseño del proceso y su relación con los atributos de calidad definidos anteriormente.

3.3.1 Métrica Tamaño Operacional de Clase

Tamaño operacional de clase (***TOC***): Número de métodos que tiene una clase. A continuación, se muestra en la tabla 8 los atributos que se evalúan al aplicar la métrica *TOC*.

Tabla 8 Tamaño operacional de clases (*TOC*)

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: elaboración propia

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

Tabla 9 Criterios de evaluación para la métrica *TOC*

Atributo	Categoría	Criterio
Responsabilidad.	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Complejidad de implementación.	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Reutilización.	Baja	>2*Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	<=Promedio

Fuente: elaboración propia

Resultados obtenidos en la aplicación de la métrica *TOC*

Los resultados obtenidos luego de aplicar las métricas *TOC* arrojan que el diseño propuesto tiene una calidad aceptable, teniendo en cuenta que el 73 % de las clases poseen una

cantidad de procedimientos menor o igual al promedio general de **59,3333**, esto conlleva a que las evaluaciones sean positivas en los atributos de calidad involucrados (responsabilidad, complejidad de implementación y reutilización), ver instrumento de evaluación de la métrica *TOC* en el Anexo 17.

En la figura 11 se muestran los resultados de la evaluación de la métrica *TOC* para los atributos antes mencionados, los cuales obtuvieron los siguientes resultados:

- **Responsabilidad**: se muestra que el 73% de las clases tienen una baja responsabilidad ya que este atributo se distribuyó equitativamente entre todas las clases del sistema. Esta característica permite que, en caso de fallos, como la responsabilidad está distribuida de forma equilibrada ningún proceso sea demasiado crítico como para dejar fuera de servicio el sistema.
- **Complejidad de implementación**: la figura muestra que la mayoría de las clases tienen una baja complejidad, este atributo está distribuido equitativamente. Esta característica permite mejorar el mantenimiento y soporte de estos procesos.
- **Reutilización**: se ve demostrado que el diseño de la solución es eficiente ya que todos los procesos tienen un alto grado de reutilización.

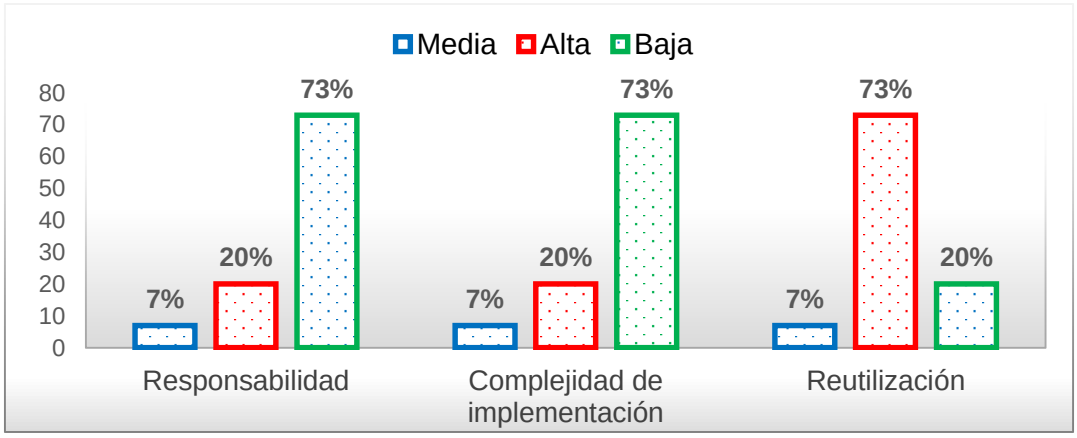


Figura 11 Resultados de la evaluación de la métrica *TOC*
Fuente: elaboración propia

3.3.2 Métrica Relaciones entre Clases

Relaciones entre clases (***RC***): Esta dado por el número de relaciones de uso de una clase con otra. A continuación, se muestra en la tabla 10 los atributos de calidad que se miden al evaluar la métrica *RC*.

Tabla 10 Relaciones entre clases (*RC*)

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: elaboración propia

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

Tabla 11 Criterios de evaluación para la métrica *RC*

Atributo	Categoría	Criterio
Acoplamiento.	Ninguno	0
	Baja	1
	Media	2
	Alta	>2
Complejidad de mantenimiento.	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Reutilización.	Baja	>2*Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	<=Promedio
Cantidad de pruebas.	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio

Fuente: elaboración propia

Resultados obtenidos en la aplicación de la métrica *RC*

Los resultados obtenidos luego de aplicar las métricas *RC* arrojan que el diseño propuesto tiene una calidad aceptable, teniendo en cuenta que el 73 % de las clases poseen una cantidad de relaciones de uso menor o igual al promedio general de **1,733333**, esto conlleva

a que las evaluaciones sean positivas en los atributos de calidad involucrados (acoplamiento, complejidad de mantenimiento, cantidad de pruebas y reutilización), ver instrumento de evaluación de la métrica *RC* en el Anexo 18.

En la figura 12 se muestran los resultados de la evaluación de la métrica *RC* para los atributos antes citados, los cuales obtuvieron los siguientes resultados:

- **Acoplamiento**: se evidencia un diseño eficiente al quedar reflejado que los procesos cuentas con un bajo acoplamiento.
- **Complejidad de mantenimiento**: queda demostrada la eficiencia de la arquitectura del sistema al evidenciarse una baja complejidad de mantenimiento.
- **Reutilización**: se ve demostrado que el diseño de la solución es eficiente ya que todos los procesos tienen un alto grado de reutilización.

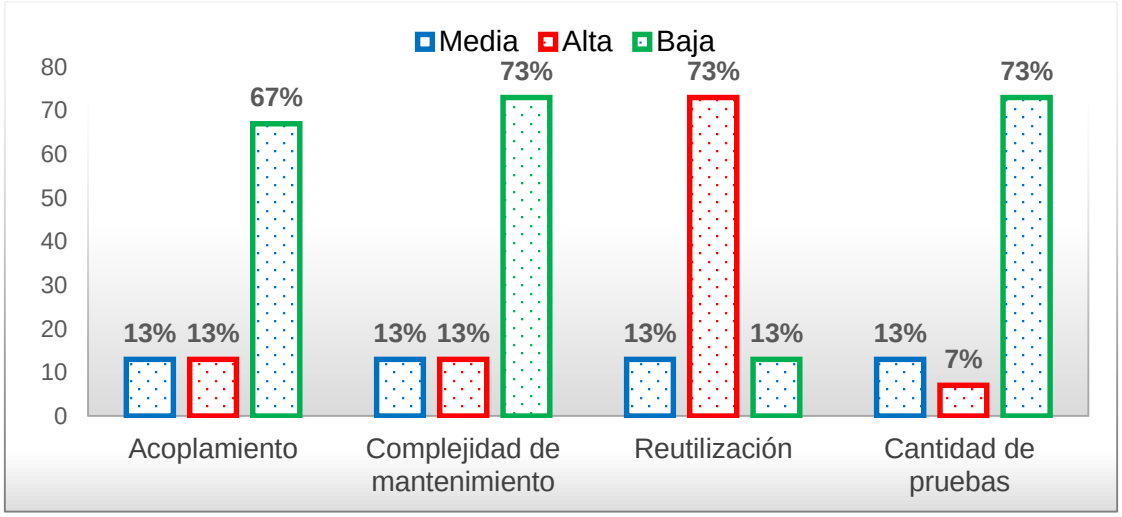


Figura 12 Resultados de la evaluación de la métrica *RC*
Fuente: elaboración propia

3.3.3 Matriz de inferencia de indicadores de calidad

La matriz inferencia de indicadores de calidad, también llamada matriz de cubrimiento, es una representación estructurada de los atributos de calidad y métricas utilizadas para evaluar la calidad del diseño propuesto. Dicha matriz permite conocer si los resultados obtenidos de las relaciones atributo/métrica es positivo o no, llevando estos resultados a una escalabilidad numérica donde, si los resultados son positivos se le asigna el valor de 1, si son negativos toma valor 0 y si no existe relación es considerada como nula y es representada con un guión simple (-). Luego se puede obtener un resultado general para cada atributo promediando todas sus relaciones no nulas.

A continuación, se muestran los resultados obtenidos.

Tabla 12 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

Tabla 13 Rango de valores para la evaluación de la relación atributo/métrica

Categoría	Rango de valores
Responsabilidad.	<=0.4
Complejidad de implementación.	>0.4 y <0.7
Reutilización.	>=0.7

Fuente: elaboración propia

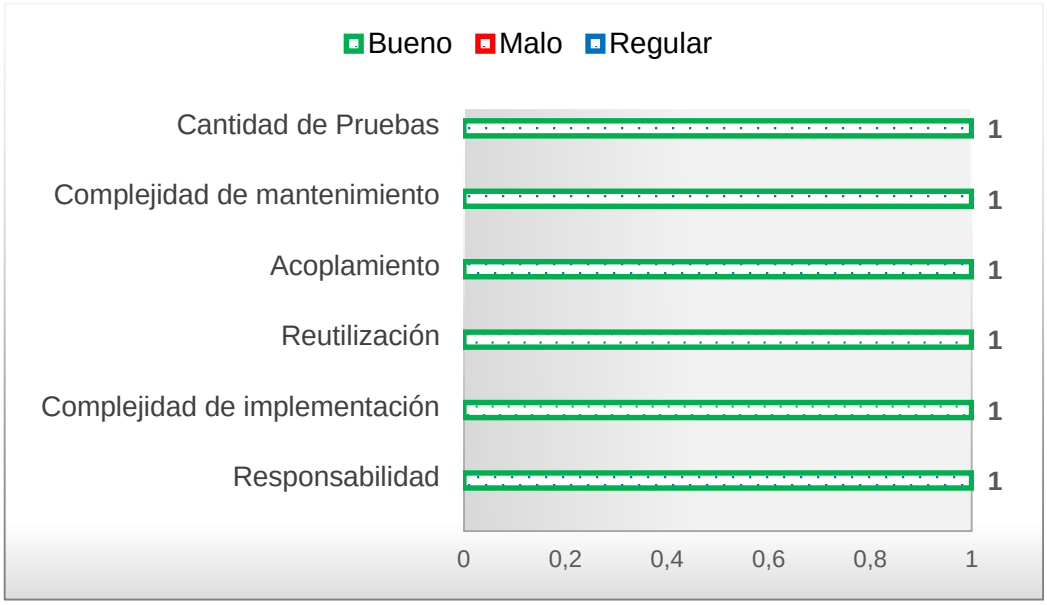


Figura 13 Atributos de calidad evaluados en las métricas
Fuente: elaboración propia

Resultados obtenidos en la aplicación de las métricas de *software*

En el proceso de evaluación realizado utilizando las métricas de *software*, se afirma que los resultados obtenidos de los atributos de calidad evaluados en las métricas aplicadas

anteriormente son positivos ya que todos los atributos de calidad mantienen un buen comportamiento.

3.4 Pruebas del *software*

Las pruebas son una actividad, en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos específicos, los resultados son observados y registrados para una posterior evaluación. Son un elemento de suma importancia para la calidad del *software*, por lo que se llevan a cabo durante todo el ciclo de vida del mismo. Constituyen una revisión final de las especificaciones del diseño y de la codificación.(55)

Para el desarrollo de las pruebas se tienen en cuenta un conjunto de estrategias a seguir, y de esta forma lograr la calidad requerida y el cumplimiento de los objetivos. La estrategia de prueba que elige la mayor parte de los equipos de *software* se ubica entre estos dos extremos. Toma un enfoque incremental de las pruebas; inicia con las pruebas de unidades individuales del programa, pasa a pruebas diseñadas para facilitar la integración de las unidades, y culmina con pruebas que se realizan sobre el sistema construido.(55)

Para alcanzar el objetivo de las pruebas de *software* es necesario planear y ejecutar una serie de pasos (pruebas de unidad, integración, validación y sistema). Las pruebas de unidad e integración se concentran en la verificación funcional de cada componente y en la incorporación de componentes en la arquitectura del *software*. La prueba de validación demuestra el cumplimiento de los requisitos del *software* y la prueba del sistema valida el *software* una vez que sea incorporado a un sistema mayor.(55)

Con el objetivo de validar la solución implementada se concibieron y ejecutaron una serie de pruebas para el proceso desarrollado. Se empleó la prueba de unicidad y aceptación.

3.4.1 Pruebas de aceptación

Las pruebas de aceptación del usuario es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el *software* está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el *software* fue construido.(55)

Caja negra

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

- Funciones incorrectas o faltantes.
- Errores de interfaz.
- Errores de estructura de datos o en acceso a base de datos externas.
- Errores de comportamiento o desempeño.
- Errores de inicialización y término.(55)

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

Método de partición equivalente

La partición equivalente es un método de prueba de caja negra que divide el campo de entrada en clases de datos que tienden a ejecutar determinadas funciones del *software* de las que pueden derivarse casos de prueba. Permite examinar los valores válidos y no válidos de las entradas existentes en el *software*. La partición equivalente se esfuerza por definir un caso de prueba que descubra ciertas clases de errores y reduce el número total de casos de prueba que deben desarrollarse.(55)

A continuación, se expone una síntesis de los resultados obtenidos al aplicar la prueba de caja negra con el uso del método partición equivalente (ver Tabla 14), este método se aplicó al **RF1**, el cual mostró resultados satisfactorios.

Tabla 14 Resultados de la prueba uno de caja negra sobre la HU 1

Caso de Prueba	
Código: SC_1	**Historia de usuario**: 1
Nombre: crear estructuras de control.	
Descripción: prueba para la funcionalidad de crear estructuras de control.	
Condiciones de Ejecución: • El usuario debe estar autenticado. • Establecer la configuración de *BD* para *Oracle* 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

De manera general para las pruebas de caja negra se realizaron dos iteraciones. En una primera iteración se detectaron dos no conformidades (NC):

- No se crearon las estructuras de control al guardar la configuración establecida.
- No se creó correctamente los *triggers* en la *BD* al guardar la configuración.

En una segunda iteración se resolvieron las dos NC detectadas en la primera iteración y no se detectaron NC.

Con la aplicación de las pruebas de aceptación se validó que el proceso cumple con el funcionamiento esperado y permitió al cliente determinar su conformidad con la solución propuesta, desde el punto de vista de las funcionalidades y rendimiento del proceso, donde se validó que el alcance es correcto.

3.4.2 Pruebas de unicidad

Las pruebas de unicidad están enfocadas a los elementos más pequeños del *software*. Aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera. La prueba de unidad está orientada a la técnica de prueba caja blanca.(55)

Caja blanca

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

- Garanticen que todas las rutas independientes dentro del módulo se han ejercitado por lo menos una vez.
- Ejerciten los lados verdaderos y falsos de todas las decisiones lógicas.

- Ejecuten todos los bucles en sus límites y dentro de sus límites operacionales.
- Ejerciten estructuras de datos internos para asegurar su validez.(55)

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

Técnica de Camino básico

Esta técnica permite obtener una medida de la complejidad lógica de un diseño procedimental y utiliza esa medida como guía para la definición de un conjunto básico de caminos de ejecución, de los cuales se obtienen los casos de prueba, que garantizan la ejecución de cada sentencia del programa al menos una vez, durante las pruebas. (55)

Para la ejecución de esta técnica existen cuatro pasos básicos:

- A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado: se utiliza para representar el flujo de control lógico de un programa.
- Se calcula la complejidad ciclomática del grafo: el valor calculado define el número de rutas independientes en el conjunto básico de un programa, y proporciona un límite superior para el número de pruebas que deben aplicarse, lo cual asegura que todas las instrucciones se hayan ejecutado por lo menos una vez.
- Se determina un conjunto básico de caminos independientes: es cualquier ruta del programa que ingresa por lo menos un nuevo conjunto de instrucciones de procesamiento o una nueva condición.
- Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico. (55)

Se muestra a continuación el caso de prueba guiado por los pasos anteriores al método "**buildingReplicableStructureGroup()**", el cual se localiza en la clase "**DBStructureChangeManager**". El objetivo de este método es construir el grupo de estructura replicable que contiene las acciones referentes a los cambios de estructuras que se van a replicar el cual se modificó para eliminar las acciones enviadas para el dialecto "**Oracle9iDialect**". Se escogió este método por la importancia que tiene para el proceso desarrollado.

Tabla 15 Resultados de la prueba de caja blanca

DBStructureChangeManager. buildingReplicableStructureGroup()	
0	`public void buildingReplicableStructureGroup() throws SQLException {`
1	`try {`

```
2   HibernateDialect dialect = null;
    dialect = DialectFactory.getDialect(sqlMetadataManager);
    int cantActions = 0;
    String cantActionstoReplics =
    DialectUtils.getCantActionstoReplics(dialect);
    Statement statement = connection.createStatement();
    ResultSet resultCount =
    statement.executeQuery(cantActionstoReplics);
3   if (resultCount.next()) {
4      cantActions = resultCount.getInt(1);
    }
5   resultCount.close();
6   while (cantActions > 0) {
7      int cantElementsToRead =
            Integer.parseInt(GeneralConfig
            .getCANT_ACTIONS_IN_GROUP());
       String valuesOfMirrorTable = DialectUtils
            .getOfMirrorTable(dialect, cantElementsToRead);
       ResultSet result = statement.executeQuery(
            valuesOfMirrorTable);
       ReplicableStructureGroup replicableStructureGroup =
       new ReplicableStructureGroup(
            getNewReplicableStructureGrupId(),
       ReplicableStructureGroup.REPLICATOR_WITH_FUNCTION,
            GeneralConfig.getNodeId());
8      while (result.next()) {
9          ReplicableStructureAction
           replicableStructureAction =
           getReplicaleStructureActionByResultSet(result);
           replicableStructureGroup
           .addSentence(replicableStructureAction);
           int id;
10         if (dialect instanceof Oracle9iDialect) {
11             id = result.getInt(RepConstants
                    .ID_STRUCTURE_CONTROL_TABLE);
           }
12         else {
               id = result.getInt(RepConstants.REPLIC_ID);
           }
13         Statement statementDelete = connection
                   .createStatement();
           String deleteRepAction = DialectUtils
                   .deleteTupleOfMirrorTable(dialect, id);
           MetaDataUtils.runDDL(statementDelete,
                   deleteRepAction);
           statementDelete.close();
```

```
        }
14      result.close();
        replicableStructureGroupStorageDao
                .persist(replicableStructureGroup);
        transactionProcessor
                .sendReplicableStructureGroup (
                replicableStructureGroup);
        ResultSet newCount = statemen
                .executeQuery(cantActionstoReplics);
15      if (newCount.next())
16         cantActions = newCount.getInt(1);
17      newCount.close();
      }
18    statement.close();
      connection.rollback();
      connection.close();
      logger.info("Finalizado proceso de construcción de
      la agrupación ");
    }
19  catch (Exception e) {
20    e.printStackTrace();
    }
21  }
```

Fuente: elaboración propia

Se procede a la construcción del grafo correspondiente al código fuente mostrado anteriormente, se utiliza la técnica de camino básico, cuyo procedimiento se ejecutó manual y automático con la utilización de la herramienta *Visustin*[41] concerniente al código fuente presentado (ver figura 14).

[41] Visustin es un programa automatizado para la creación de diagramas de flujo para desarrolladores de *software* y escritores de documentos. Usa la ingeniería inversa en su código fuente para crear diagramas de flujo o diagramas de actividad UML. 56. ABBAS, A. S. Re *engineering of legacy software systems*. 2015, nº

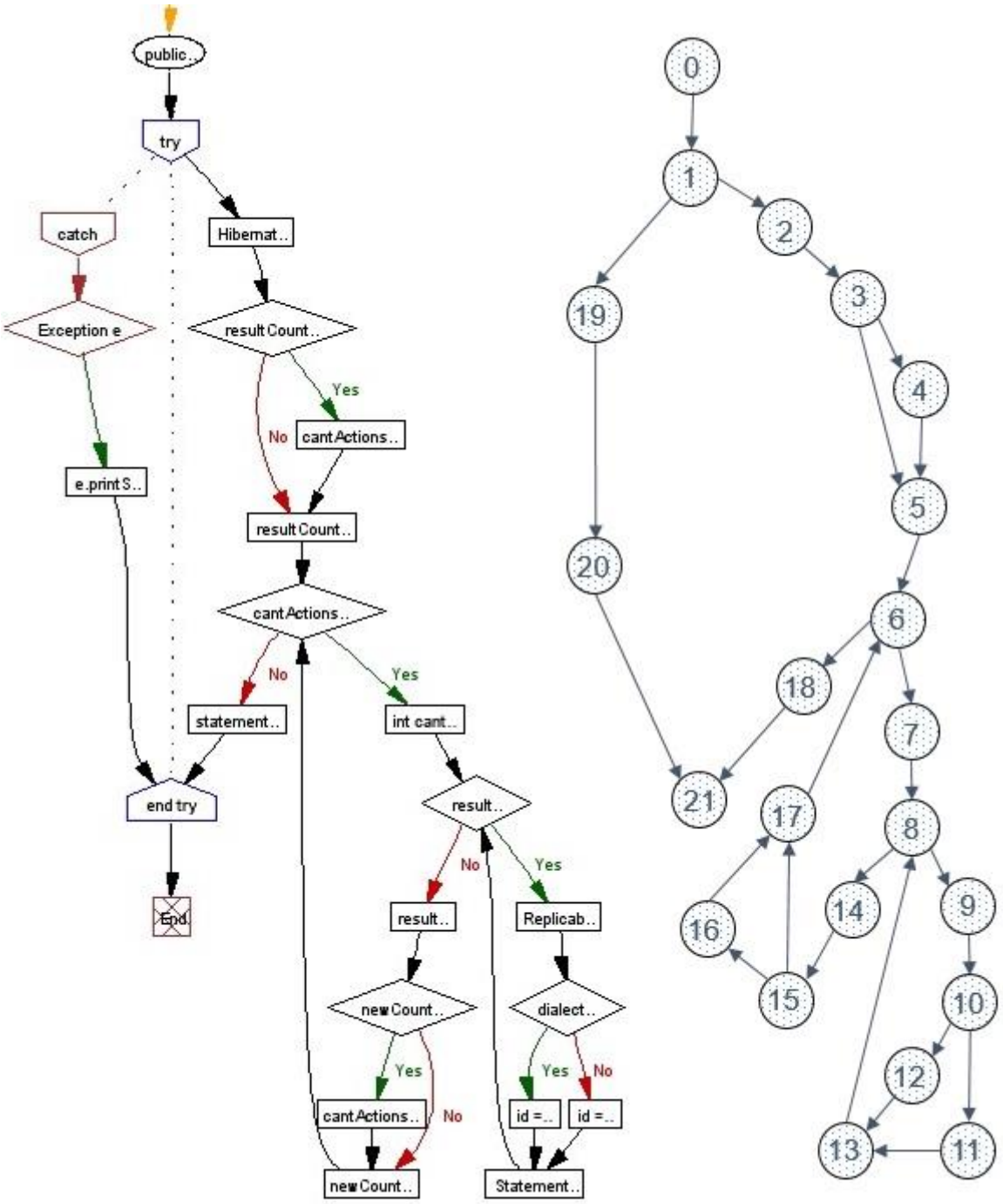


Figura 14 Flujo automático y manual correspondiente al método buildingReplicableStructureGroup()
Fuente: elaboración propia

Luego de haber construido el grafo se realiza el cálculo de la complejidad ciclomática mediante las tres fórmulas descritas a continuación, las cuales deben arrojar el mismo resultado para asegurar que el cálculo de la complejidad es correcto (ver tabla 15).

Tabla 16 Complejidad ciclomática correspondiente al método buildingReplicableStructureGroup()

DBStructureChangeManager. buildingReplicableStructureGroup()		
V(G) = R	**V(G) = A - N + 2**	**V(G) = P + 1**
R = 7 (regiones del grafo**)**	**A = 27 (**aristas**)** **N = 22 (**nodos**)**	**P = 6 (**nodos predicados**)**
	V(G) = 27 - 22 + 2	**V(G) = 6 + 1**
V(G) = 7	**V(G) = 7**	**V(G) = 7**

Fuente: elaboración propia

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

Se representan a continuación los caminos básicos:

- **Ruta 1**: **{**0, 1, 19, 20, 21**}**
- **Ruta 2**: **{**0, 1, 2, 3, 5, 6, 18, 21**}**
- **Ruta 3**: **{**0, 1, 2, 3, 4, 5, 6, 7, 8, 14, 15, 16, 17, 6, 18, 21**}**
- **Ruta 4**: **{**0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 13, 8, 14, 15, 17, 6, 18, 21**}**
- **Ruta 5**: **{**0, 1, 2, 3, 4, 5, 6, 7, 8, 14, 15, 17, 6, 18, 21**}**
- **Ruta 6**: **{**0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 13, 8, 14, 15, 17, 6, 18, 21**}**
- **Ruta 7**: **{**0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 8, 14, 15, 16, 17, 6, 18, 21**}**

A continuación, se plantean los casos de prueba que resguardan los caminos independientes presentados.

Tabla 17 Casos de pruebas de los caminos independientes

No	Entrada	Resultado
1	Error de conexión con la *BD* al obtener la cantidad de acciones a replicar.	Retorna una excepción.
2	La tabla espejo está vacía, no hay acciones para replicar.	No construye la agrupación.
3	No obtiene los valores de la tabla de control.	Recorre la tabla espejo pero no construye la acción replicable.

4	Luego de construir la primera agrupación se interrumpe la conexión a la *BD*.	Retorna una excepción.
5	Error de conexión.	Envía y persiste un grupo de estructura replicable vacío.
6	Error de conexión con la *BD* al obtener la cantidad de acciones a replicar nuevamente para continuar ciclo.	Retorna una excepción.
7	Obtiene los valores de la tabla espejo, accede al esquema de la *BD* para obtener la tabla y construir las *RSA*.	Construye y ordena el envío del *RSG*.

Fuente: elaboración propia

Resultados de las pruebas de caja blanca

Las pruebas de caja blanca se aplicaron al código fuente del proceso, donde se comprobó el buen funcionamiento de los **RF** (**RF1**, **RF2**, **RF3**, **RF4** y **RF5**), lográndose las respuestas deseadas, estas se pudieron corroborar con la ayuda de las métricas de *software*.

De manera general para las pruebas de caja blanca se ejecutaron tres iteraciones. En una primera iteración al recibir un *RSG* con 10 acciones, se detectaron cinco NC, dos de creación, dos de modificación y una de eliminación:

- No especifica correctamente el id, cuando se va a eliminar la acción capturada en la tabla de control una vez construido un *RSG*.
- No estaba correcta la consulta *SQL* al aplicar las acciones de creación de un esquema.
- No es correcto el tipo de dato al aplicar la acción crear una columna.
- No se especifica correctamente el propietario de la tabla en la consulta *SQL* al aplicar la acción crear una tabla.

En una segunda iteración se resolvieron las cinco NC detectadas en la primera iteración. En esta iteración se recibió un *RSG* con 10 acciones, se detectaron tres NC, una de eliminación y dos de actualización:

- No se creaba correctamente el *SQL* al crear una tabla con una columna.
- No era reconocido por el gestor el tipo de dato al adicionar una columna.
- No se creaba correctamente el *SQL* al eliminar un esquema.

En una tercera iteración se resolvieron las tres NC detectadas en la segunda iteración y se recibió un *RSG* con 10 acciones, no se detectaron NC.

En las pruebas de caja blanca realizadas al sistema se detectaron un total de 8 no conformidades. La Figura 15 muestra una gráfica donde se refleja la cantidad de no conformidades detectadas por iteración. El sistema fue liberado en la tercera iteración.

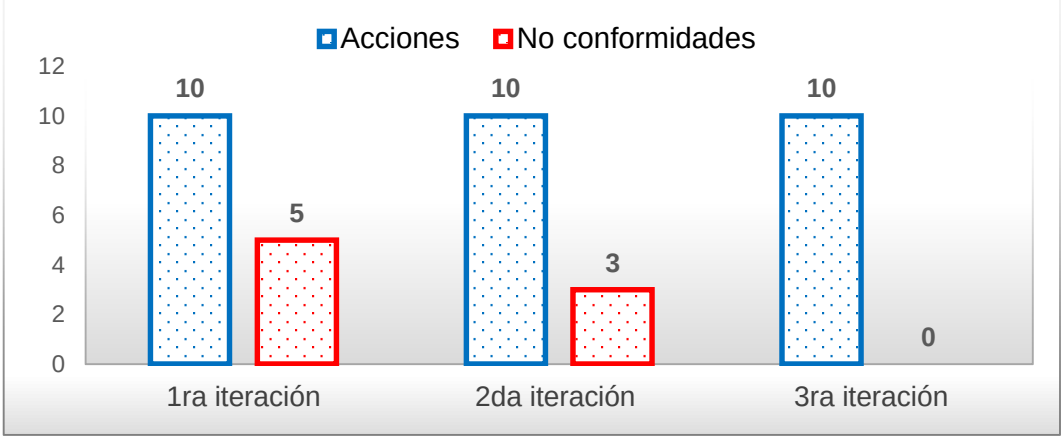


Figura 15 Cantidad de no conformidades
Fuente: elaboración propia

3.5 Resultados obtenidos

El proceso de implementación y pruebas a lo largo de la investigación arrojó los siguientes resultados:

- Se solucionaron los problemas planteados, obteniéndose un correcto funcionamiento del sistema.
- Los resultados obtenidos mediante las pruebas realizadas produjeron resultados satisfactorios con el cumplimiento de los requisitos definidos.

3.6 Consideraciones del capitulo

- En este capítulo se realizaron las tareas de ingeniería, se llevó a cabo la implementación del sistema y la descripción de la solución desarrollada.
- Se logró documentar las no conformidades mediante las pruebas realizadas y se corrigieron los errores de ejecución de los procesos no satisfactorios.

CONCLUSIONES GENERALES

Conforme al cumplimiento del objetivo y las tareas propuestas para la realización del proceso desarrollado se puede llegar a las siguientes conclusiones:

- El empleo de la metodología, herramientas y tecnologías definidas por el proyecto al cual se realiza el proceso permitió evitar problemas de compatibilidad al ser integrado al Replicador de datos Reko.
- El proceso desarrollado aseguró actualizar automáticamente los cambios realizados sobre la estructura de una *BD Oracle* garantizando la coherencia en los datos que se replican con el Replicador de datos Reko.
- Con las pruebas realizadas se logró validar la coherencia de los datos en el proceso de réplica de estructuras, por lo cual se demuestra la solidez de la implementación de la solución propuesta.

RECOMENDACIONES

Los resultados alcanzados luego del desarrollo del presente trabajo satisfacen los requerimientos definidos. No obstante, para el desarrollo de futuras versiones se recomienda:

- Agregar al sistema la posibilidad de réplica de estructura para los dialectos *MySQL* y *Microsoft SQL Server*.

REFERENCIAS BIBLIOGRÁFICAS

1. VALDÉS, D. P. *¿Qué son las bases de datos?* [Consultado el: 08/02 de 2017]. Disponible en: http://www.maestrosdelweb.com/que-son-las-bases-de-datos/.

2. ING. EDISVEL MELO ESTEVEZ , I. A. G. S., ING. YANISLEYDIS RODRÍGUEZ TAMAYO. *CONFIGURACIÓN Y MONITORIZACIÓN DE LA RÉPLICA DE ESTRUCTURA PARA EL REPLICADOR DE DATOS REKO* Cuba : Universidad de las Ciencias Informáticas.: [Consultado el: 08/02 de 2017]. Disponible en: https://veme-cuba.eventos.uci.cu/sites/default/files/public/ponencia/ponencia/node/Uciencia665.pdf.

3. SANTIESTEBAN, I. A. G. *CONFIGURACIÓN Y MONITORIZACIÓN DE LA RÉPLICA DE ESTRUCTURA PARA EL REPLICADOR DE DATOS REKO | VEME-CUBA. VEME-CUBA* [Consultado el: 08/02 de 2017]. Disponible en: https://veme-cuba.eventos.uci.cu/es/content/configuraci%C3%B3n-y-monitorizaci%C3%B3n-de-la-r%C3%A9plica-de-estructura-para-el-replicador-de-datos-reko.

4. VALDÉS, D. P. *¿Qué son las bases de datos?* Maestros del Web by Platzi: [Consultado el: 08/02 de 2017]. Disponible en: http://www.maestrosdelweb.com/que-son-las-bases-de-datos/.

5. MANNINO., M. V. *Introducción a las base de datos según Korth. Scrid* [Consultado el: 08/02 de 2017]. Disponible en: https://es.scribd.com/doc/30859379/Introduccion-a-las-base-de-datos-segun-Korth.

6. FUENTES, M. D. C. G. *MATERIAL DIDÁCTICO NOTAS DEL CURSO BASES DE DATOS* Primera edición 2013. ed. UNIVERSIDAD AUTONOMA METROPOLITANA, México D.F.: Universidad Autónoma Metropolitana, Unidad Cuajimalpa, Disponible en: http://www.cua.uam.mx/pdfs/conoce/libroselec/Notas_del_curso_Bases_de_Datos.pdf. ISBN 978-607-477-880-9.

7. DÍAZ, M. C. A. G. BASES DE DATOS DISTRIBUIDAS MIS 515. nº [Consultado el: 08/02/2017]. Disponible en: http://cursos.aiu.edu/Base%20de%20Datos%20Distribuidas/pdf/Tema%201.pdf.

8. VELASCO, J. E. P. *Esquema de base de datos, Instituto Tecnológico de Veracruz* [Consultado el: 08/02 de 2017]. Disponible en: http://www.prograweb.com.mx/tallerBD/0102Esquema.php.

9. OCAMPO, J. A. *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.* CEPAL: 2004. ISBN 9789213222836.

10. ESPAÑA., I. N. D. E. D. *Gestión orientada a asegurar la calidad de los datos en los institutos nacionales de estadística* Santiago de Chile: [Consultado el: 08/02 de 2017]. Disponible en: http://www.cepal.org/deype/ceacepal/documentos2/espana_ine.pdf.

11. ORACLE. *Database Replication* [Consultado el: 08/02 de 2017]. Disponible en: http://docs.oracle.com/cd/A64702_01/doc/server.805/a58227/ch_repli.htm.

12. CHAVEZ., O. *Administracion De Base De Datos: Replicacion.* [Consultado el: 08/02 de 2017]. Disponible en: http://chavez-atienzo-2013.blogspot.com/2013/04/replicacion.html.

13. URBANO, R. Oracle Database Advanced Replication, 11g Release 2. 2013, nº [Consultado el: 08/02/2017]. Disponible en: http://docs.oracle.com/cd/E11882_01/server.112/e10706.pdf.

14. BUCHILLÓN, V. R. *Replicación de Datos* [Consultado el: 08/02 de 2017]. Disponible en: http://www.monografias.com/trabajos82/replicaciondatos/replicaciondatos2.shtml#queeslarea.

15. *Lenguaje de Manipulación de Datos (DML)*. [Consultado el: 08/02 de 2017]. Disponible en: http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro14/53__lenguaje_de_manipulacin_de_datos_dml.html.

16. *Lenguaje de Definición de Datos (DDL)*. [Consultado el: 08/02 de 2017]. Disponible en: http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro14/52_lenguaje_de_definicin_de_datos_ddl.html.

17. PAGANINI, D. J. M. GESTION POR PROCESOS nº [Consultado el: 08/02/2017]. Disponible en: http://www.cenas.org.ar/DOC%203.pdf.

18. SOFTWARE, H. *DBMoto Cloud Edition generalidades* [Consultado el: 10/06 de 2017]. Disponible en: http://www.hitsw.com/localized/spanish/products_services/dbmoto/dbmoto_cloud/DBMoto_Cloud_Edition.html.

19. YERANDY MANSO GUERRA, ENRIQUE JOSÉ ALTUNA, , ADRIÁN GARCÍA SÁNCHEZ , YOANDY PÉREZ CÁCERES. Experiencias en el uso del symmetric para la replicación de datos en la plataforma educativa ZERA. nº [Consultado el: 10/02/2017]. Disponible en: http://scielo.sld.cu/pdf/rcci/v9n4/rcci14415.pdf.

20. HENSON, C. [Consultado el: 05/04 de 2017]. Disponible en: http://symmetricds.codehaus.org/.

21. LONG, E. *Sync Schema (DDL) Changes.* [Consultado el: 27/02 de 2017]. Disponible en: http://www.symmetricds.org/docs/how-to/sync-schema-ddl-changes.

22. JEREZ., G. R. L. *Componente para el envío y recepción de datos del proceso de réplica de estructura para el software de replicación Reko.* Ingeniero en Ciencias Informáticas, Universidad de Ciencias Informáticas (UCI), 2013.

23. RÍOS, M. N. D. L. *Manual de usuario Replicador de datos REKO.* 2015. vol. 4.0, 97 p.

24. PRESSMAN, R. S. *Ingeniería del Software, Un enfoque práctico.* Sexta ed. 2005. ISBN 84-481-3214-9.

25. SÁNCHEZ, T. R. *Metodología de desarrollo para la Actividad productiva de la UCI.* 1.2 ed. Universidad de las Ciencias Informáticas, [Consultado el: 11/06/2017].

26. FLORES, E. *Metodologías Agiles Proceso Unificado Ágil (AUP)* [Consultado el: 10/02 de 2017]. Disponible en: http://ingenieriadesoftware.mex.tl/63758_AUP.html.

27. JAMES RUMBAUGH, I. J., GRADY BOOCH. *El Lenguaje Unificado de Modelado Manual de Referencia.* Edición en Español. Nuñez de balboa, 120 28006 Madrid : PEARSON EDUCACIÓN, S.A. ed. 2000. ISBN 84-7829-037-0.

28. *VISUAL-PARADIGM. Software Design Tools for Agile Teams, with UML, BPMN and More.* [Consultado el: 10/02 de 2017]. Disponible en: https://www.visual-paradigm.com/.

29. RIVERA, F. L. O. *Introducción a la Programación en Java. Un enfoque Práctico.* 1ra ed. septiembre de 2007. ISBN 978-958-98314-8-9.

30. *ORACLE. Java EE - Java Platform*. [Consultado el: 27/02 de 2017]. Disponible en: https://www.oracle.com/java/technologies/java-ee.html.

31. GALLARDO, D. *Iniciándose en la plataforma Eclipse* [Consultado el: 14/03 de 2017]. Disponible en: https://www.ibm.com/developerworks/ssa/library/os-ecov/.

32. FUNDATION, T. A. S. *Apache ActiveMQ* [Consultado el: 10/02 de 2017]. Disponible en: http://activemq.apache.org/index.html

33. FUNDATION, T. A. S. *Apache Tomcat* [Consultado el: 10/02 de 2017]. Disponible en: http://tomcat.apache.org

34. FARAONI, F. J. G. *DESARROLLO DE UNA APLICACIÓN WEB CON SPRING FRAMEWORK PARA UN GESTOR DE UN RECETARIO* [Consultado el: 10/02 de 2017]. Disponible en: http://oa.upm.es/38731/1/TFG_Federico_Gutierrez_Faraoni.pdf.

35. *Microsoft Word - Que es Oracle*. [Consultado el: 10/02 de 2017]. Disponible en: https://iessanvicente.com/colaboraciones/oracle.pdf.

36. LARMAN, C. *MODELO DEL DOMINIO, Extraído de: UML y Patrones. 2ª Edición. Craig Larman. Prentice Hall. 2003* Disponible en: http://is.ls.fi.upm.es/docencia/is2/documentacion/ModeloDominio.pdf.

37. SOMMERVILLE, I. *Requerimientos. .* septima ed. Madrid: Pearson Education S.A: 2005. vol. 7, 108 p. Ingeniería del Software ISBN 84-7829-074-5.

38. PRESSMAN, R. S. *Ingenieria del Software, Un Enfoque Práctico.* 2005. ISBN 8448132149

39. GONZALEZ, A. *Sistemas de llamada y retorno* 2 de Junio de 2016, [Consultado el: 10/03 de 2017]. Disponible en: https://prezi.com/wyylmlq8jpyo/sistemas-de-llamada-y-retorno/.

40. SZYPERSKI, C. *Component Software: Beyond Object-oriented Programming* [Consultado el: 10/03 de 2017]. Disponible en: https://books.google.es/books/about/Component_software.html?hl=es&id=ZKlQAAAAMAAJ.

41. PELAEZ, J. C. *Arquitectura basada en Componentes.* [Consultado el: 10/03/2017]. vol. 2017, Disponible en: http://geeks.ms/jkpelaez/2009/04/18/arquitectura-basada-en-componentes/.

42. KICILLOF, C. R. N. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft* UNIVERSIDAD DE BUENOS AIRES: Disponible en: http://carlosreynoso.com.ar/archivos/arquitectura/Estilos.PDF.

43. ANTONIO, E. G. *Introducción a los Patrones de Diseño, ¿Qué son los patrones de Diseño?* [Consultado el: 11/03 de 2017]. Disponible en: http://codecriticon.com/introduccion-patrones-diseno/.

44. LARMAN, C. *UML y Patrones. Introducción al análisis y diseño orientado a objetos y al proceso unificado* Segunda edición. ed. PEARSON EDUCACIÓN,S.A, Madrid, 2003: [Consultado el: 11/03 de 2017]. Disponible en: http://www.fmonje.com/UTN/ADES%20-%20208/UML%20y%20Patrones%20%202da%20Edicion.pdf. ISBN 84-205-3438-2.

45. CARLOS A. GUERRERO, J. M. S., LUZ E. GUTIÉRREZ. *Patrones de Diseño GOF (The Gang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web , Información Tecnológica Vol. 24 (3), 103-114 (2013)* Disponible en: http://www.scielo.cl/pdf/infotec/v24n3/art12.pdf. ISBN 0718-0764.

46. ÁLVAREZ, C. *Patrones de Diseño (Active Record vs DAO)* [Consultado el: 11/03 de 2017]. Disponible en: https://www.genbetadev.com/java-j2ee/patrones-de-diseno-active-record-vs-dao.

47. IVAR JACOBSON, G. B., JAMES RUMBAUGH. *El proceso unificado de desarrollo de software.* Traducido por: Luis Joyanes, E. P. Editado por: Reading, A. W. Español ed. 2000. vol. 7, 464 p. ISBN 84-7829-036-2.

48. VELEZ, C. *Diagrama de paquetes (UML), Transcripción de Diagrama de paquetes (UML)* 6 de Mayo de 2015 [Consultado el: 10/03 de 2017]. Disponible en: https://prezi.com/utbnaeaajpo1/diagrama-de-paquetes-uml/.

49. ALCOCER PULUPA, D. E. Y. O. T., F. R. . Diseño e implementación de un prototipo de un sistema computarizado distribuido orientado al control de la utilización de los servicios en un campus universitario a nivel local con capacidad para 2000 a 5000 estudiantes. QUITO/EPN/2013. 2013, nº

50. GARCÍA LIRA, K. G. D., AILEC Y TEJERA HERNANDEZ , DAYANA CARIDAD. Conferencia #1: Continuación de la Disciplina Análisis y Diseño. En Ingenieria de software II. Departamento de ISW, Universidad de las Ciencias Informáticas, Curso: 2010 - 2011. p. 56. nº

51. Tema II. Fase de Construcción. En Conferencia 6. Disciplina de Implementación. Cuba, Universidad de las Ciencias Informáticas. 2011, nº

52. *Definición de Código fuente , General Tecnología- Definista* [Consultado el: 09/05 de 2017]. Disponible en: http://conceptodefinicion.de/codigo-fuente/.

53. CALLEJA, M. A. Carmen. Estándares de codificación. nº [Consultado el: 19/05/2017]. Disponible en: http://www.cisiad.uned.es/carmen/estilo-codificacion.pdf.

54. MOLPECERES, A. Convenciones de código para el lenguaje de programación Java. 2001, nº [Consultado el: 19/05/2017]. Disponible en: http://www.um.es/docencia/vjimenez/ficheros/practicas/ConvencionesCodigoJava.pdf.

55. PRESSMAN, R. S. *Estrategia de prueba del software. En Ingenieria del Software, Un Enfoque Práctico.* 6 ed. España: Editorial McGraw-Hil, 2002. 382 p. ISBN 8448132149