Universidad de las Ciencias Informáticas

Facultad 5

Centro de Consultoría y Desarrollo de Arquitecturas Empresariales

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

Título: Módulo para la resolución de conflictos generados durante el proceso de réplica de estructura para el Replicador de datos REKO.

Autores:

Liset Martínez Almaguer

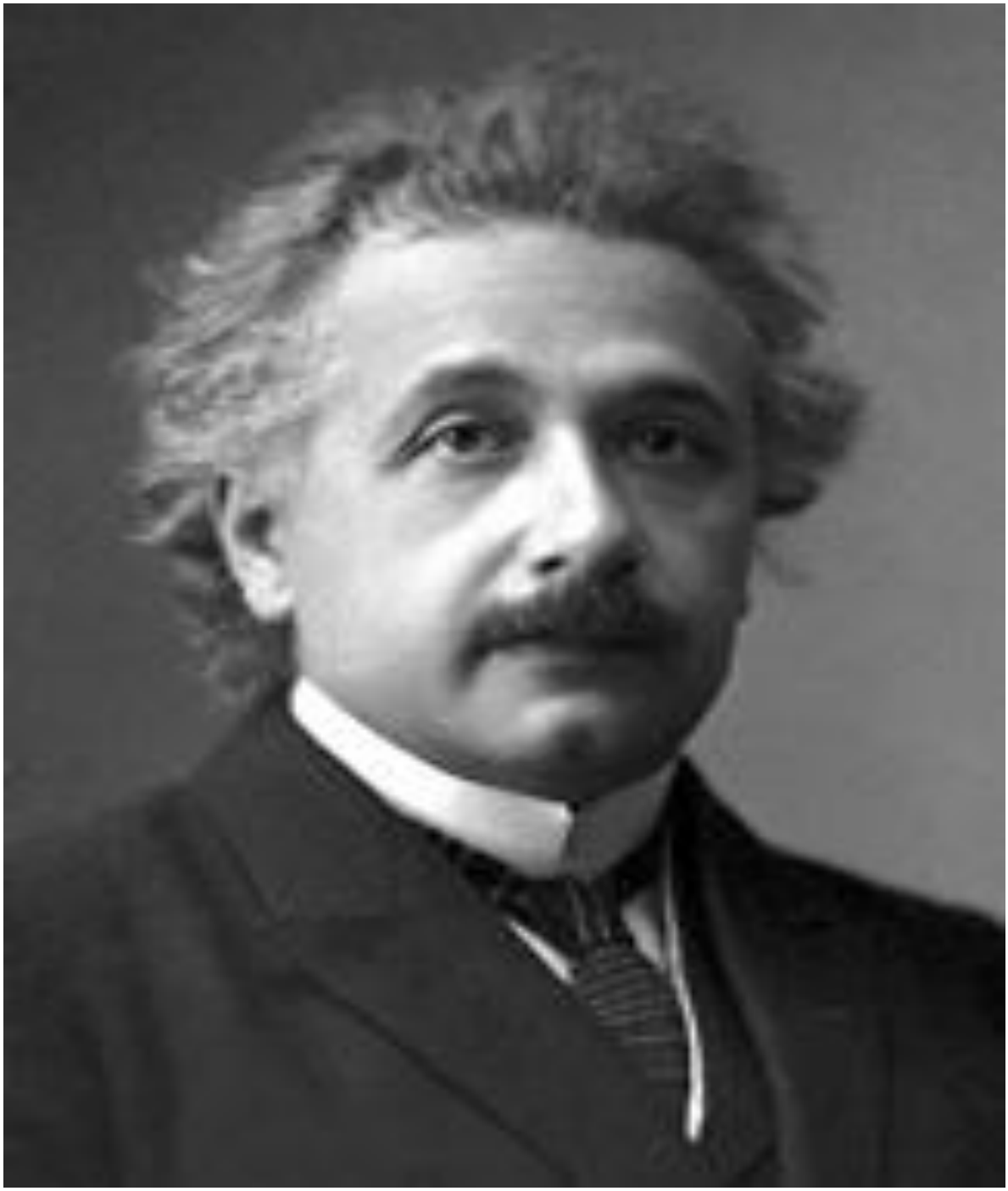
Osvaldo Cuba López

Tutora:

Ing. Gloria Raquel Leyva Jerez

La Habana, Junio de 2015

"Año 57 de la Revolución"



"Si buscas resultados distintos no hagas siempre lo mismo."

Albert Einstein.

De Liset

Agradezco a mi familia, en especial a mi mamá que ella me ha dado todo en la vida, por su amor y dedicación.

A mi papá por apoyarme y estar siempre pendiente de mí.

A Ermis y Marbelis por dejarme entrar en sus vidas y apoyarme en todo momento.

A Eric por creer en mí y estar a mi lado en los buenos y malos momentos, por apoyarme y ayudarme todos estos años, por tener paciencia con mis malcriadeces, gracias mi amor, te amo.

A mi compañero de tesis y amigo Osvaldo, por confiar en mí, por su dedicación y abnegación todo este tiempo.

A mi tutora Gloria Raquel por ser tan exigente y guiarme por los caminos correctos en la investigación, por su confianza y por estar apoyándome en todo momento.

A Edilberto por acogerme como otro miembro más de su familia.

A Javier, Yanet, Claudia Celeste, Mirnerys, Claudia María, Marlen, Miguely y Kiki por su amistad y el apoyo incondicional que me ofrecieron en los buenos y malos momentos.

A mis compañeros de aula, desde primer año hasta quinto, en especial a Arisney.

A todas aquellas personas que de una forma u otra me ayudaron y me ofrecieron su apoyo.

De Osvaldo

Agradezco a mi familia, en especial a mi mamá que ella me ha dado todo en la vida, por su amor y dedicación.

A mi papá por apoyarme y estar siempre pendiente de mí.

A los profesores que contribuyeron a mi formación y son parte de este resultado Yisel Neris, Yadira, Claudia, Yausel y Zaida.

A mi tutora Gloria Raquel por ser tan exigente y guiarme por los caminos correctos en la investigación, por su confianza y por estar apoyándome en todo momento.

A mis compañeros de aula, desde primer año hasta quinto, en especial a Liset, Kiki, Rioger, Walter, Felipe, Lorgio, Mirnerys y Orson.

De Liset

Dedico esta investigación a mi mamá, que la amo mucho y gracias a ella he realizado todos mis sueños.

A mi papá que siempre ha estado a mi lado y a Ermis por apoyarme en todo.

A Eric por ser mi compañero, mi amigo, mi novio, mi amor, mi todo, por compartir su vida todos estos años conmigo, por apoyarme en los momentos difíciles y por todos los días del mundo que nos quedan por vivir.

A mi compañero de tesis Osvaldo.

De Osvaldo

Dedico esta investigación a mi mamá, que la amo mucho y gracias a ella he realizado todos mis sueños.

DECLARACIÓN JURADA DE AUTORÍA

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

Para que así conste firmamos la presente a los ____ días del mes de _____ del año 2015.

Firma de los autores:

_____ _____

Liset Martínez Almaguer. Osvaldo Cuba López.

Firma de la tutora:

_____

Ing. Gloria Raquel Leyva Jerez.

DATOS DE CONTACTO

Autores

Liset Martínez Almaguer

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

Correo: lmalmaguer@estudiantes.uci.cu

Osvaldo Cuba López

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

Correo: ocuba@estudiantes.uci.cu

Tutora

Ing. Gloria Raquel Leyva Jerez

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

Correo: grleyva@uci.cu

RESUMEN

La creciente informatización de los procesos aumenta el uso de sistemas de base de datos distribuidos e impulsa el desarrollo y uso de herramientas de réplica para mantener actualizados los objetos de las base de datos. En la Universidad de las Ciencias Informáticas se desarrolla el Replicador de datos REKO el cual detecta y resuelve los conflictos generados durante la aplicación de los cambios del proceso de réplica y sincronización de datos. Sin embargo, no procesa los conflictos de la réplica de estructura y afecta la coherencia de los datos que intervienen en este proceso, debido a que se introducen errores tales como: tablas o datos incompletos, problemas de integridad referencial e inconsistencias entre los sistemas de base de datos que se encuentran distribuidos.

El presente trabajo de diploma propone una solución para los conflictos generados en el proceso de réplica de estructura, que permite eliminar los errores introducidos durante la aplicación de cambios de estructura con el Replicador de datos REKO para evitar incoherencias entre los datos que se replican.

Para el desarrollo de la solución se describe el problema inicial que dio origen al desarrollo del módulo propuesto, el uso de métodos teóricos y empíricos, también aspectos fundamentales sobre soluciones homólogas de replicadores de datos a nivel mundial que se pusieron en práctica en la solución propuesta. Se hizo uso de la metodología AUP que permite describir el diseño e implementación del módulo, el cual fue desarrollado con herramientas *Open Source* y bibliotecas de clases con licencias gratuitas.

Palabras claves: conflictos, réplica de estructura, módulo.

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

La creciente informatización de los procesos, propicia la descentralización geográfica de las organizaciones y el aumento de la necesidad del empleo de Sistemas de Base de Datos Distribuidos (SBDD). Cada SBDD está compuesto por nodos de una red de computadoras, que permiten realizar procesamiento autónomo y ejecutar acciones locales o globales, posibilita que el acceso y procesamiento a los datos sea más rápido. El uso de estos sistemas impulsa el desarrollo y empleo de herramientas de réplica de datos para mantener actualizados los objetos de la Base de Datos (BD). Los replicadores de datos facilitan la recuperación, aumentan la tolerancia a fallos y fiabilidad en la distribución de los datos.(1)

En la Universidad de las Ciencias Informáticas (UCI) específicamente en el Centro de Consultoría y Desarrollo de Arquitecturas Empresariales (CDAE), se cuenta con el Replicador de datos REKO, como una solución de réplica en ambientes distribuidos. El Replicador de datos REKO tiene como objetivo brindar una herramienta multiplataforma que le permite al usuario, con el menor esfuerzo, cubrir las necesidades fundamentales de réplica, tales como: sincronización, transferencia de datos entre diversas localizaciones, 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.(1)

El Replicador de datos REKO cuenta con el módulo de Conflictos que permite la resolución automática y manual de los conflictos generados durante la réplica de datos en ambientes distribuidos, con conexión y sin conexión. Este módulo es capaz de detectar los conflictos ocurridos al actualizar los cambios aplicados durante la réplica y sincronización de datos, brinda la posibilidad al usuario de definir la forma de resolver los conflictos en cada instancia y especifica la resolución automática o manual. No obstante, si durante la aplicación de cambios de estructura se genera un conflicto, la réplica se detiene automáticamente y afecta el aprovechamiento laboral del personal involucrado en el proceso de réplica, debido a que se dedica más tiempo y esfuerzo para resolver las dificultades de forma manual. El software ignora

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.

la presencia de dichos conflictos, lo que ocasiona inconsistencias entre los datos que se replican, tablas o datos incompletos, e incluso problemas de integridad referencial, durante el proceso de réplica, todo esto afecta la coherencia de los datos.

Por la situación antes expuesta, se identifica el siguiente **problema a resolver:**
¿Cómo eliminar los errores introducidos durante la aplicación de cambios de estructura, para evitar incoherencias en los datos que se replican con el Replicador de datos REKO?

Este problema se enmarca en el **objeto de estudio** referente al proceso de aplicación de cambios de estructura en sistemas de base de datos distribuidos, donde el **campo de acción** comprende las soluciones para resolver conflictos de estructura con el Replicador de datos REKO.

Para resolver el problema identificado se propone el siguiente **objetivo general:**
Desarrollar un módulo para la resolución de conflictos generados durante el proceso de réplica de estructura, que permita evitar incoherencias en los datos que se replican con el Replicador de datos REKO.

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

- Establecimiento de los fundamentos teórico-metodológicos del proceso para la resolución de conflictos en la réplica de estructura.
- Caracterización del proceso de réplica de estructura en el Replicador de datos REKO.
- Realización del diseño del módulo para la resolución de conflictos.
- Implementación del módulo para la resolución de conflictos en el proceso de réplica de estructura en el Replicador de datos REKO.
- Validación de la contribución lograda a través de la introducción del módulo para la resolución de conflictos en el proceso de réplica de estructura en el Replicador de datos REKO, para valorar la calidad del producto implementado.

La **idea a defender** en la presente investigación, se plantea de la siguiente manera:
Con el uso del módulo para la resolución de conflictos del proceso de réplica de estructura, se eliminarán errores introducidos durante la aplicación de cambios de estructura, evitando incoherencias en los datos que se replican con el Replicador de datos REKO.

Para el desarrollo de la investigación se utilizaron **métodos científicos**, todos bajo la concepción dialéctica-materialista como método general.

Los **métodos teóricos** empleados se enuncian 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.
- **Análisis - Síntesis e Inductivo – Deductivo:** 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 módulo para la resolución de conflictos en el proceso de réplica de estructura para el Replicador de datos REKO.
- **Modelación:** permitió generar los artefactos necesarios y diseñar el módulo para la resolución de conflictos de estructura.

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.

Con la presente investigación se espera como resultado que el Replicador de datos REKO cuente con un módulo para la resolución de conflictos generados durante el proceso de réplica de estructura, que permita evitar incoherencias de los datos que se replican y brindar 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 módulo. 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 módulo para demostrar su robustez.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

El presente capítulo aborda 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 solución del problema científico y define la metodología, herramientas y lenguaje a utilizar en la solución propuesta.

1.1 Marco conceptual

1.1.1 Base de Datos

El autor Manuel Sierra[2] plantea:

"Una base de datos es un sistema informático a modo de almacén. En este almacén se guardan grandes volúmenes de información. Permite automatizar el acceso a la información y poder acceder a esta de forma rápida y fácil, además de poder efectuar cambios de manera más eficiente".(2)

Rosa M. Mato[3] declara:

"… una base de datos es un conjunto de datos interrelacionados, almacenados con carácter más o menos permanente en la computadora, o sea, una colección de datos variables en el tiempo".(3)

David Bolton[4] expone:

"Una base de datos es una aplicación que administra datos y permite rápido almacenamiento y recuperación de los mismos".(4)

En correspondencia con los autores citados anteriormente, se puede sintetizar que una BD se caracteriza por:

Un conjunto de información persistente, perteneciente al mismo contexto y almacenada de forma organizada. Se puede acceder a los datos fácilmente y ejecutar cambios de manera eficiente. Están constituidas principalmente por tablas, relaciones y funciones.

1.1.2 Base de Datos Distribuida

El autor Fredy Carranza[5] considera que:

2 Manuel Sierra, analista y programador informático con experiencia en la tecnología JAVA, PHP y C#.

3 Rosa M. Matos, Licenciada en Cibernética Matemática, Profesora Auxiliar, del Centro de Estudios de Ingeniería de Sistemas (CEIS), del Instituto Superior Politécnico José Antonio Echeverría (ISPJAE), Ciudad de La Habana

4 David Bolton, *Bachelor of Science* (Licenciado en Ciencias) y desarrollador de software en Londres, Inglaterra.

5 Fredy Carranza Athó, Ingeniero Informático, Licenciado y Máster en Ciencias de la Computación.

"*Una Base de Datos Distribuida 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*". (5)

El equipo de desarrollo asume dicho concepto porque lo considera válido y acorde a los objetivos de la investigación. Las ideas expuestas por este autor coinciden con las emitidas por otros autores, tales como: J.C. Date[6] y Abraham Silberschatz[7].

En las Base de Datos Distribuida (BDD), las BD se distribuyen en forma de nodos como se muestra en la Figura 1, donde estos pueden ser agregados fácilmente, con independencia y autonomía entre ellos. La probabilidad de que un nodo afecte el funcionamiento del sistema es baja.

Figura 1 Interconexión de un sistema de BDD
Fuente: elaboración propia del equipo de desarrollo

1.1.3 Esquema o estructura de una BD

El colectivo de la especialidad de BD del Instituto Tecnológico de Veracruz emite que:
"*El esquema o estructura de una BD se describe teniendo en cuenta el tipo de gestor. De forma general, define las tablas, campos de cada tabla y las relaciones entre cada campo y cada*

[6] C. J Date, conferencista, investigador y consultor independiente, especializado en la tecnología de bases de datos.
[7] Abraham Silberschatz, Director de la Universidad estatal de *Nueva York* en *Stony Brook*, Vicepresidente del Centro de Investigación de Ciencias de la Información en los Laboratorios Bell y miembro de las sociedades ACM e IEEE.

tabla, de un Sistema Administrador de Base de datos. Generalmente en la práctica el término esquema de la base de datos se refiere al diseño físico de la base de datos ".(6)

Los autores de la presente investigación asumen la definición anteriormente citada, porque se considera que la estructura de una BD define las tablas, campos de cada tabla y las relaciones entre cada campo y cada tabla, refiriéndose al diseño físico de la BD.

1.1.4 Coherencia de los datos

Para la autora Dharma Rojas[8] la coherencia de los datos:

"…está referida a la idoneidad de los datos para ser cambiados de diferentes maneras y para diferentes usos…"(7)

José Antonio Ocampo[9] expresa que:

"…es la idoneidad de los datos para ser combinados en forma fiable de diferentes maneras y distintos usos, procedan de una fuente única o investigaciones estadísticas de diversas naturalezas"(8)

En la presente investigación se asumen las posiciones de los autores anteriormente citados y consideran que la coherencia de los datos en el contexto de réplica se entiende como la correspondencia entre los datos que se envían desde un nodo local y se aplican en un nodo remoto para mantener la integridad de los datos.

1.1.5 Réplica

Para el autor Marius Cristian Mazilu[10] representa:

"*El proceso de intercambio de información para garantizar la coherencia entre los recursos redundantes, tales como: componentes de software o hardware. Permite mejorar la confiabilidad, tolerancia a fallos y accesibilidad.*"(9)

El autor Randy Urbano[11] emite que:

"*La réplica en el contexto de los Sistemas de Gestión de Bases de Datos, se entiende como el proceso de copiar y mantener objetos de una base de datos, entiéndase tablas, secuencias y/o*

8 Dharma Rojas, Universidad de Los Andes (Venezuela), Administración y contaduría

9 José Antonio Ocampo, Secretario General de las Naciones Unidas para Asuntos Económicos y Sociales, y ex Secretario Ejecutivo de la CEPAL.

10 Marius Cristian Mazilu, cibernético, Máster en el soporte de base de datos. Sus ámbitos de desarrollo son: desarrollo y gestión de aplicaciones de base de datos, desarrollo de aplicaciones w eb para empresas y marketing w eb.

11 Randy Urbano, escritor técnico de Oracle.

triggers[12]*, 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".*(10)

Para un grupo de desarrolladores de Microsoft:
"*La réplica es un conjunto de tecnologías destinadas a la copia y distribución de datos y objetos de base de datos desde una base de datos a otra, para luego sincronizar ambas bases de datos y mantener su coherencia. La replicación permite distribuir datos entre diferentes ubicaciones y entre usuarios remotos o móviles mediante redes locales y de área extensa*".(11)

El equipo de desarrollo considera utilizar en la investigación la definición de réplica emitida por Randy Urbano debido a que es la más ajustada a la temática abordada.

Un software de réplica realiza la réplica de datos. Bajo las circunstancias anteriormente descritas permite la actualización de las BD que se encuentran distribuidas, producto a la ejecución de comandos DML[13]. Mientras que, un replicador de estructura o un replicador de datos que realiza réplica de estructura es aquel que aplica los cambios producidos por la ejecución de comandos DDL[14].

1.1.6 Lenguaje de Manipulación de Datos

El Lenguaje de Manipulación de Datos se conoce genéricamente como DML. Se especializa en la utilización de la base de datos, es decir, permite la consulta y mantenimiento de la base de datos.(12)
Al realizar consultas en las bases de datos se emplean los siguientes comandos:

- *Select*: utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.
- *Insert*: utilizado para cargar lotes de datos en la base de datos en una única operación.
- *Delete*: utilizado para eliminar registros de una tabla de una base de datos.
- *Update*: utilizado para modificar los valores de los campos y registros especificados.(13)

1.1.7 Lenguaje de Definición de Datos

El Lenguaje de Definición de Datos también se conoce como DDL. Se especializa en la escritura de esquemas, es decir, en la descripción de la base de datos. Define como el sistema

[12] Conocido como desencadenador. Es un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación DML.
[13] Lenguaje de Manipulación de Datos, del inglés Data Manipulation Language.
[14] Lenguaje de Definición de Datos, del inglés Data Definition Language.

organiza internamente los datos. Incluye órdenes para modificar, borrar o definir las tablas en las que se almacenan los datos de la base de datos.(12)

Las operaciones básicas que se utilizan son:

- *Create*: para crear nuevas tablas, campos e índices.
- *Drop*: para eliminar tablas e índices.
- *Alter*: para modificar las tablas al agregar campos o cambiar la definición de los campos.(13)

1.1.8 Conflictos en la réplica

Para Jofman Pérez Tarancón[15] se entiende como conflicto en la réplica a:

"…cualquier irregularidad originada en el proceso de aplicación de la información que impide que la transacción pueda ser aplicada. Los conflictos normalmente son asociados a inconsistencia en la información existente en un ambiente con relación al otro". (14)

El equipo de desarrollo considera que la definición expuesta se relaciona con los conflictos tanto de datos como de estructura. Por ende, se asume que los conflictos de estructura se refieren a los errores generados durante la réplica de estructura.

1.1.9 Módulo

Para el autor Cory Janssen[16] :

"Un módulo es un componente de software. A menudo se puede utilizar en una variedad de aplicaciones y funciones con otros componentes del sistema. Funciones similares se agrupan en la misma unidad de código de programación y funciones se desarrollan como unidades de código para que el código puede ser reutilizado por otras aplicaciones".(15)

El autor Leandro Alegsa[17] emite que:

"En programación, un módulo es un software que agrupa un conjunto de subprogramas y estructuras de datos. Los módulos son unidades que pueden ser compiladas por separado y los hace reusables y permite que múltiples programadores trabajen en diferentes módulos en forma simultánea, produciendo ahorro en los tiempos de desarrollo."(16)

Los autores de la presente investigación consideran que en programación, un módulo es un software que agrupa un conjunto de subprogramas y estructuras de datos que poseen las

[15] Jofman Pérez Tarancón: Master en Informática Aplicada de la UCI, La Habana, Cuba.
[16] Cory Janssen es un co-fundador de *Janalta Interactive Inc*., la compañía matriz de *Techopedia*.
[17] Leandro Alegsa, Trabaja en la empresa Alegsa y es profesor de informática de la Universidad Tecnológica de Santa Fe, Argentina

propiedades de ocultamiento de la información, cohesión y acoplamiento. Los módulos son unidades que permiten a múltiples programadores trabajar en diferentes módulos en forma simultánea, produciendo ahorro en los tiempos de desarrollo.

1.2 Replicadores

Producto a la necesidad de mantener la información actualizada en diferentes áreas geográficas las empresas comercializan soluciones de réplica en el mercado de software, muchas de ellas incluyen en el software la réplica y otras como soluciones que se pueden obtener de forma independiente. Se realizó un estudio de replicadores en busca de una propuesta para la resolución de conflictos en el proceso de réplica de estructura para el Replicador de datos REKO.

1.2.1 SymmetricsDS

Es un software de código abierto, con soporte para la réplica multi-maestro, la sincronización filtrada, y la transformación a través de la red en un entorno heterogéneo, aplicación desarrollada en Java. Es compatible con múltiples suscriptores con una dirección o, réplica de datos asíncrona bidireccional. Utiliza tecnologías web y base de datos para replicar los datos como una operación en tiempo real. El software fue diseñado para escalar a un gran número de nodos, trabajar a través de las conexiones de bajo ancho de banda, y soportar períodos de interrupción de la red. Funciona con la mayoría de sistemas operativos, sistemas de archivos y BD y soporta a los gestores MySQL, Oracle, Microsoft SQL Server, PostgreSQL, DB2, Informix, Interbase, Firebird, HSQLDB, H2, Apache Derby y Greenplum.

En la versión 3.0 se le incluye la detección y resolución de conflictos. El cual está basado en tres aspectos fundamentales:

- El usuario en la configuración puede previamente definir cómo actúa el sistema frente a distintos tipos de conflictos.
- En caso de que se detecte un conflicto y el usuario no haya definido previamente una solución, el sistema automáticamente, en dependencia del tipo de acción: inserción, actualización o eliminación, lo resuelve de manera predeterminada o lo ignora.
- Resolución manual: en caso que durante la réplica de datos se generen conflictos se debe obtener una solución a través de una interfaz.

A partir de la versión 3.1.9 incluye la agrupación del trabajo de depuración y mejoras en el rendimiento de los datos de encaminamiento para un gran número de nodos al replicar

esquema de base de datos en los nodos remotos y sincroniza los cambios de esquema para las tablas cuando se alteran.(17)

1.2.2 Slony o Slony-I

Sistema de réplica maestro-esclavo. Es una poderosa herramienta utilizada para guardar los datos, recuperarse ante fallos, optimiza el procesamiento de transacciones en línea, la descarga de informes, copia de seguridad y restauración. Es una solución de réplica basada en activadores que se replican de forma asincrónica. La réplica puede ser en cascada. En lugar de recibir datos replicados directamente desde el maestro, un esclavo puede recibir datos replicados de otro esclavo. (18)

Ofrece réplica con la utilización de disparadores para recolectar los cambios ocurridos en la base de datos. Su modo normal de operación implica que todos los nodos deben estar disponibles todo el tiempo. Sólo soporta el servidor de base de datos PostgreSQL, cuya réplica es sólo entre bases de datos con estructuras iguales. No tiene la capacidad de replicar objetos de gran tamaño. Para la resolución de conflictos establece que se deben resolver automáticamente según las políticas que están establecidas y en caso que no exista una política para un conflicto determinado se detiene el proceso de réplica hasta que manualmente se resuelva el problema.(19)

1.2.3 Daffodil Replicator

Es una herramienta de código abierto desarrollada en Java para la integración, migración y protección de datos en tiempo real. Permite bidireccionar datos de réplica y la sincronización entre base de datos homogéneas y heterogéneas, soporta los gestores de bases de datos Oracle, MySQL, Daffodil DB y PostgreSQL. Presenta la capacidad de detectar y permitir la resolución de conflictos, es independiente de la plataforma, provee soporte para la réplica de datos de gran tamaño y réplica entre base de datos con estructuras iguales, no es capaz de replicar en un escenario multi-maestro.

Daffodil Replicator es capaz de detectar y resolver los conflictos que ocurren entre el publicador y el suscriptor durante la réplica de datos. Clasifica los conflictos en tres tipos: unicidad, eliminación y actualización. La forma de solución depende de quién tiene la información confiable sea el publicador o el suscriptor y dónde se detecte el conflicto.(20)

1.2.4 Oracle Streams

Basa su funcionamiento en mensajes, unidad básica de la información que comparte. Estos mensajes son colocados dentro de un flujo de datos que permite propagar la información dentro de la misma base de datos y hacia otras. Los mensajes pueden ser originados, transformados y/o enviados por el gestor de Oracle u otras herramientas externas que interactúen con Oracle Streams, brinda una alta flexibilidad y variadas funcionalidades. La unidad básica de captura de cambios se denomina *Logical Change Records* (por sus siglas en inglés, LCR), de la cual existen dos tipos *row* LCR, relativa a los cambios por operaciones DML o DDL LCR, relativa a las operaciones de DDL, como su nombre lo indica.

Oracle Streams establece mecanismos de detección de conflictos que se generan durante la réplica e incorpora rutinas para la resolución automática de potenciales conflictos. Además, permite que los usuarios creen sus propias rutinas y empleen reglas que satisfagan las necesidades de su negocio. Los conflictos que no se puedan filtrar por los mecanismos antes mencionados se almacenan en registros para un posterior tratamiento manual.(21)

1.2.5 Microsoft SQL Server Developer Network (MSDN)

Replicador de Microsoft SQL Server que admite una amplia gama de cambios de esquemas en objetos publicados.(22)

Su utilidad en sistemas de desarrollo está limitada bajo licencia, además como es una herramienta de Microsoft SQL Server, la réplica está definida únicamente para sistemas operativos de Microsoft. El comportamiento de la detección y resolución de conflictos depende de las opciones que brinda MSDN para este caso y según la elección se determina qué puede ser un conflicto y qué tipo de solución se puede aplicar. La réplica de mezcla ofrece solucionadores automáticos que se clasifican en 4 tipos (solucionador de conflictos predeterminado basado en prioridad, controlador de lógica de negocios, solucionador personalizado proporcionado por Microsoft) y un solucionador interactivo que proporciona una interfaz de usuario donde permite elegir manualmente la forma en que se solucionan los conflictos en tiempo de ejecución. Incluye componentes y una Interfaz de Programación de Aplicaciones, del inglés *Application Programming Interface* (API) intuitiva y flexible que facilitan la sincronización entre Microsoft SQL Server, SQL Server Express, SQL Server Compact, y bases de datos de SQL Azure. (23)

1.2.6 Replicador de datos REKO

El Replicador de datos REKO constituye una solución práctica y de bajos esfuerzos de configuración a las principales necesidades de réplica. Permite que dos BD puedan replicar entre sí aunque se encuentren en subredes distintas y utilicen protocolos de transmisión diferentes, la única restricción es que todos los nodos deben estar conectados a un servidor central JMS[18] o en caso de un *clúster*[19] de servidores JMS. Con el empleo de herramientas libres y la metodología de desarrollo AUP. (24)

Su desarrollo sobre la plataforma JEE[20] 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* Spring. (25)

La administración y configuración de la réplica se realiza a través de una interfaz web, por lo que es administrable vía remota con la utilización de un navegador.

Principales funcionalidades que brinda el software REKO en su versión 4.0:

- Soporta la réplica de transacciones a través de *Hibernate*[21] y la réplica de acciones a través de *triggers*.
- Soporte para réplica de datos en ambientes con conexión y sin conexión.
- Soporte para réplica entre BD con diferentes estructuras.
- Soporte para réplica de datos entre los gestores PostgreSQL, Oracle, MySQL y Microsoft SQL Server.
- Soporte para resolución de conflictos de datos automática y manual.(1)

La resolución automática o manual de conflictos de datos se define de la siguiente manera:

- Conflicto de unicidad: sucede cuando la réplica de un registro intenta violar una restricción de integridad, ya sea por llave primaria o única.
- Conflicto de actualización: un conflicto de actualización ocurre cuando una transacción intenta actualizar un registro que no existe.
- Conflicto de eliminación: un conflicto de eliminación ocurre cuando una transacción intenta borrar o actualizar un registro que no existe o que no se puede borrar un elemento por dependencia de llave foránea en cascada.

[18] Servicio de Mensajería de Java, del inglés *Java Message Service.*
[19] 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.
[20] Plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java, del inglés *Java Enterprise Edition.*
[21] Herramienta de mapeo objeto/relacional para entornos Java.

- Conflictos desconocidos: a partir de la primera ocurrencia de un conflicto desconocido se puede automatizar la opción que se elija de forma manual, específicamente para este tipo de conflicto. Se asegura que cuando ocurra nuevamente este tipo de conflicto se le aplica la acción automatizada elegida inicialmente.

En la resolución de los conflictos se regula el comportamiento de los nodos involucrados en el escenario de réplica respecto al nodo local, estos nodos pueden tener dos comportamientos (Ganador o No Ganador).(26)

1.2.7 Resultados de la investigación

Las soluciones estudiadas anteriormente no cumplen con los requerimientos básicos planteados, debido a que las herramientas libres por sí solas no cuentan con un módulo o solución que pueda ser integrado al Replicador de datos REKO para resolver los conflictos que se generan cuando se realizan cambios de estructura en la BD. No se tiene el código de la implementación en las fuentes consultadas. En algunos casos se hace alusión a una pequeña síntesis de los pasos o acciones a ejecutar para resolver los conflictos de datos, manuales o automáticos. Además la integración de algunos de los replicadores que posean esta funcionalidad en el Replicador de datos REKO disminuiría considerablemente el rendimiento del sistema. En las soluciones privativas no se esclarece totalmente como se realiza dicho proceso y al contar con una solución sería muy costoso su uso.

Por 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.

El funcionamiento de las soluciones estudiadas anteriormente puede ser usado para resolver los conflictos generados durante el proceso de réplica de estructura. Se aportan elementos básicos para la solución, tales como la configuración que se debe realizar previamente a los conflictos generados, los pasos a seguir para la resolución de los conflictos tanto manual como automática, se tiene en cuenta la información que prevalece en los nodos involucrados en el conflicto y la clasificación de los conflictos.

1.3 Metodología de desarrollo

Las metodologías para el desarrollo del software 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 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". (27)

Para guiar el proceso de desarrollo se empleará la metodología AUP adaptada al ciclo de vida definido por la UCI para la actividad productiva, debido a que es la metodología que ha designado la universidad para los proyectos de desarrollo y el módulo a desarrollar será integrado en la próxima versión del proyecto. Por tanto, el flujo de trabajo y artefactos generados en este trabajo deben corresponderse al desarrollo del proyecto.

Proceso Unificado Ágil

El Proceso Unificado Ágil o *Agile Unified Process* (por sus siglas en inglés, AUP[22]) es una versión simplificada del Proceso Unificado de Rational o *Rational Unified Process* (por sus siglas en inglés, RUP). Este describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio con el uso de técnicas ágiles y conceptos que aún se mantienen válidos en RUP.

El AUP aplica técnicas ágiles que incluyen:

- Desarrollo Dirigido por Pruebas (*Test Driven Development* - TDD).
- Modelado Ágil.
- Gestión de Cambios Ágil.
- Refactorización de Base de Datos para mejorar la productividad.

El proceso unificado es un marco de desarrollo de software iterativo e incremental. A menudo es considerado como un proceso altamente ceremonioso porque especifica muchas actividades y artefactos involucrados en el desarrollo de un proyecto de software. Dado que es un marco de procesos, puede ser adaptado y la más conocida es RUP de IBM[23].(28)

Al igual que en RUP, en AUP se establecen cuatro fases, sin embargo en la adaptación para la actividad productiva se definieron las fases de Inicio, Ejecución y Cierre, las cuáles definen las actividades a realizar como se muestra en la siguiente tabla:

Tabla 1 Fases de AUP adaptadas a la actividad productiva de la universidad

Fases	Objetivos de las fases
Inicio	Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un

[22] Versión simplificada de RUP, que favorece el desarrollo ágil de proyectos.
23 *International Business Machines Corp*

	estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
Ejecución	En esta fase se ejecutan las actividades requeridas para desarrollar el software, se incluye el ajuste de los planes del proyecto y considera 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. Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente. Además, en la transición se capacita a los usuarios finales sobre la utilización del software.
Cierre	En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Fuente: Metodología de desarrollo para la actividad productiva de la UCI (29)

Además AUP propone siete disciplinas y para los proyectos de la UCI se definen ocho disciplinas como se puede ver en el Anexo 1.

Entre las técnicas ágiles que utiliza AUP se encuentra el modelado ágil que permite encapsular los requisitos funcionales en Historias de usuario (HU), Descripción de requisitos por procesos o por Casos de uso. El proyecto Replicador de datos REKO en su versión 4.0 realiza la descripción de las HU. Por tanto, se hará uso de esta técnica para encapsular los requisitos funcionales.

AUP define nueve roles y se decide para el ciclo de vida de los proyectos de la UCI proponer los once roles que a continuación se muestran:

Tabla 2 Roles y artefactos

Roles	Artefactos de salida
Jefe de proyecto **Planificador**	Plan de desarrollo de software. Método de estimación. 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 Administrador de BD	Modelo de diseño. Arquitectura de software.

Fuente: Metodología de desarrollo para la actividad productiva de la UCI (29)

1.4 Herramientas y tecnologías

En este epígrafe se tratan los conceptos relacionados con las herramientas y tecnologías del entorno de réplica del Replicador de datos REKO que han sido empleadas desde versiones anteriores, las cuales aún son utilizadas debido a que disminuye la curva de aprendizaje y el tiempo de desarrollo. Teniendo en cuenta que al utilizar otras herramientas, cuando se realice la integración del módulo para la resolución de conflictos podrían existir problemas de compatibilidad.

Lenguaje Unificado de Modelado 2.0 (por sus siglas en inglés, UML[24])

Es un lenguaje estándar en el análisis y diseño de sistemas de cómputo que permite especificar, visualizar, construir y documentar todos los elementos que forman un sistema de software, desde una perspectiva orientada a objetos. Este modelado visual es independiente del lenguaje de implementación, los diseños realizados con el uso de UML se pueden implementar en cualquier lenguaje que soporte las posibilidades de UML, principalmente lenguajes orientados a objetos.(30)

Visual Paradigm para UML 8.0

Esta es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software. Permite visualizar todos los diagramas de clases, código inverso y generar

[24] Unified Modeling Language

documentación. Cuenta con la ventaja de ser una herramienta multiplataforma y permite la integración con Eclipse IDE[25]. (31)

Presenta licencia gratuita y comercial. Es fácil de instalar y actualizar, y es compatible entre ediciones. Genera código para Java y exportación como HTML[26].(32)

Java

Lenguaje de programación[27] orientado a objeto, de código abierto e independiente de la plataforma. Se distingue por ser 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. Todas estas características lo califican como un lenguaje de programación robusto.(33)

Eclipse STS[28] 3.6.1

El IDE *Spring Tool Suite* (por sus siglas en inglés, STS) es una herramienta libre desarrollada por Spring Source, basada actualmente en Eclipse 3.x, para construir aplicaciones empresariales enriquecidas por los proyectos de Spring Source. Una de las principales razones para emplear STS es que incluye herramientas para el desarrollo en lenguaje Java, para Spring.(34)

Java Empresarial 1.7 (por sus siglas en inglés, JEE[29])

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[30] 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.(35)

Conectividad de base de datos Java 4.0 (por sus siglas en inglés JDBC[31])

API que permite la ejecución de operaciones sobre base de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual accede y utiliza el dialecto SQL del modelo de base de datos.(36)

---

[25] 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.

[26] Lenguaje de Marcado de Hipertexto del inglés HyperText Markup Language

[27] Idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras.

[28] Del inglés *Spring Tool Suite.*

[29] Java Empresarial, del inglés *Java Enterprse Edition*

[30] Interfaz de programación de aplicaciones, del inglés *Application Programming Interface.*

[31] Conectividad de base de datos Java, del inglés *Java Database Connectivity*

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.(37)

Apache Tomcat 7.0

Apache Tomcat es un *contenedor* de *servlet*[32] 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.(38)

Spring 2.0

Contenedor[33] y *framework*[34] 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.(39)

Servicio de Mensajería Java 1.0

La API *Java Message Service* (JMS por sus siglas en inglés) es el estándar de mensajería que permite a los componentes de aplicaciones basados en la plataforma JEE crear, enviar, recibir y leer mensajes. Hace posible la comunicación confiable de manera síncrona y asíncrona.(40)

JUnit 4.4

Framework de código abierto para diseñar, construir y ejecutar pruebas unitarias de aplicaciones Java. Se integra con muchos IDEs como Eclipse. Produce varios tipos de reportes a partir de los resultados obtenidos en la ejecución de las pruebas. Soporta la jerarquía entre las pruebas que permiten establecer la prioridad y el orden de unas con otras.(41)

Dojo 1.2.0

Es un *framework* de código abierto desarrollado en JavaScript destinado a facilitar el rápido desarrollo de JavaScript o de las aplicaciones basadas en Ajax y sitios web. Su idea es la de abstraer al desarrollador de las complejidades del HTML y de las discrepancias existentes entre navegadores, que hacen que el código JavaScript a utilizar sea diferente. Potencia el desarrollo del lado del cliente en una aplicación web.(42)

[32] Programa Java que se ejecuta dentro de un servidor Web. Reciben y atienden las solicitudes de los clientes web.
[33] Interfaz entre el componente y la plataforma sobre la que se ejecuta. Facilita los servicios que éste necesita para su funcionamiento.
[34] Arquitectura reusable que brinda la estructura genérica y de comportamiento para un grupo de abstracciones de software.

PostgreSQL 9.1

Gestor de BD orientadas a objetos, muy conocido y usado en entornos de software libre. Puede funcionar en múltiples plataformas. Cuenta con un rico conjunto de tipos de datos, que permiten además su extensión mediante tipos y operadores definidos y programados por el usuario. Su administración se basa en usuarios y privilegios. Es altamente confiable en cuanto a estabilidad se refiere. Puede extenderse con bibliotecas externas para soportar encriptación.(43)

1.5 Consideraciones del capítulo

A partir del análisis realizado en este capítulo es posible concluir que:

- Para la solución del problema no es viable el uso total o parcial de herramientas de réplica ya desarrolladas para la resolución de conflictos de estructura.
- Se exponen y justifican las diferentes herramientas y metodología que se usarán en el desarrollo de la aplicación, las cuales guiarán el proceso de desarrollo. Emplear la metodología AUP y las mismas herramientas utilizadas para el desarrollo del Replicador de datos REKO en versiones anteriores, disminuye la curva de aprendizaje y el tiempo de desarrollo.
- El desarrollo de un módulo para la resolución de conflictos generados durante el proceso de réplica de estructura, permitirá al Replicador de datos REKO, dar solución a dichos conflictos para actualizar automáticamente los cambios realizados sobre la estructura de una base de datos en ambientes distribuidos.

CAPÍTULO 2. PROPUESTA DE SOLUCIÓN

Este capítulo especifica los requisitos funcionales y no funcionales, describe las historias de usuarios, los patrones arquitectónicos, patrones de diseño aplicados, los diagramas de clases del diseño y el modelo de despliegue. El diseño y caracterización del sistema se realizará con la referencia de los conceptos y términos definidos en el capítulo anterior y la metodología AUP.

2.1 Descripción del proceso a automatizar

El Replicador de datos Reko permite realizar configuraciones de réplica, enviar, recibir y sincronizar los datos en ambientes con conexión y sin conexión, así como la réplica de estructura, utiliza el gestor de BD PostgreSQL. Durante el proceso de réplica y sincronización de estructura al aplicar los cambios sobre la BD destino pueden ocurrir conflictos de estructura, lo que afecta la coherencia de los datos. El sistema no es capaz de gestionar manual o automáticamente los conflictos de estructura que se generan.

Una vez desarrollado el módulo para la resolución de conflictos de estructura se podrán brindar variantes de soluciones manuales y automáticas para los distintos tipos de conflictos de estructura que puedan ocurrir durante el proceso de réplica de estructura y evita incoherencia en los datos que se replican. Las variantes establecen procedimientos que están enfocados a las posibles vías de solución que se establezcan en cada instancia.

2.2 Modelo de dominio

Para descomponer el dominio del problema hay que identificar los conceptos, los atributos y las asociaciones del dominio que se juzgan importantes. El resultado puede expresarse en un modelo de dominio o modelo conceptual.(44)

2.2.1 Diagramas de clases del dominio

Con el objetivo de describir y expresar el contexto en que se desarrolla el módulo se exponen los conceptos que son significativos para la presente investigación, los cuales se encuentran en la siguiente figura.

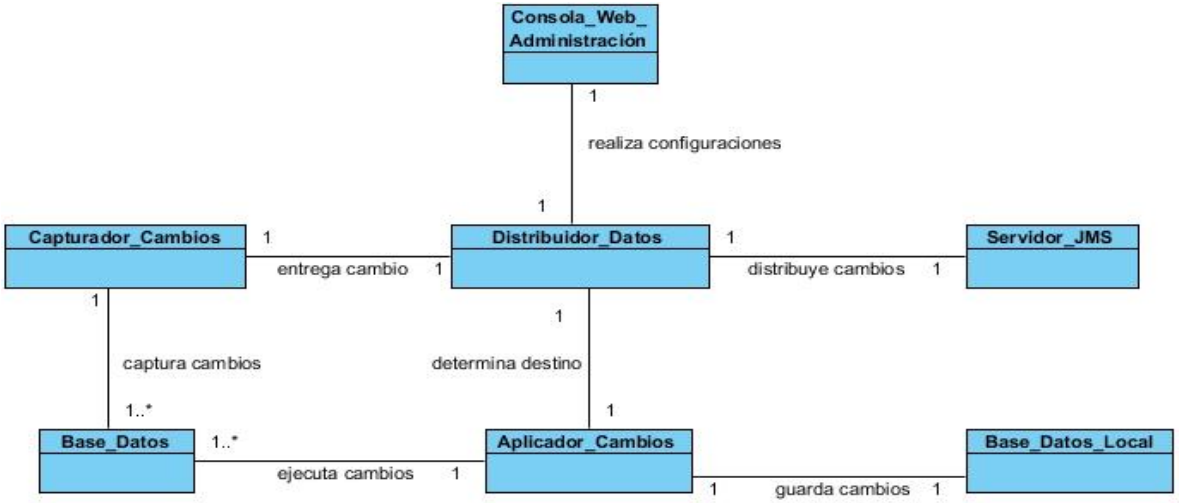


Figura 2 Modelo de dominio
Fuente: elaboración propia del equipo de desarrollo

2.2.2 Descripción de las clases del modelo de dominio

Para una mejor comprensión del modelo de dominio del módulo, se proporciona una breve descripción de las clases que lo integran, las cuales son:

- 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.
- Capturador_Cambios: se encarga de capturar los cambios realizados a la Base_Datos y de entregarlos al Distribuidor_Datos.
- Distribuidor_Datos: determina el destino de cada configuración de cambio, se responsabiliza de su llegada.
- 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 son la descripción de los servicios proporcionados por el sistema y sus restricciones operativas. Los 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 que permite descubrir, analizar, documentar y verificar esos servicios y restricciones se denomina ingeniería de requisitos.(45)

2.3.1 Requisitos funcionales

Los requisitos funcionales de un sistema describen las capacidades o funciones que el sistema debe cumplir, los servicios que de él se esperan, o los que proveerá.(45)

El levantamiento de requisitos para el módulo propuesto arrojó como requisitos funcionales los siguientes:

RF1 Guardar la configuración para la resolución de conflictos de estructura.

RF2 Capturar los conflictos de estructura generados.

RF3 Resolver automáticamente los conflictos de creación de la estructura para un nodo ganador.

RF4 Resolver automáticamente los conflictos de creación de la estructura para un nodo no ganador.

RF5 Resolver automáticamente los conflictos de modificación de la estructura para un nodo ganador.

RF6 Resolver automáticamente los conflictos de modificación de la estructura para un nodo no ganador.

RF7 Resolver automáticamente los conflictos de no existencia de la estructura para un nodo ganador.

RF8 Resolver automáticamente los conflictos de no existencia de la estructura para un nodo no ganador.

RF9 Resolver manualmente los conflictos de creación de la estructura para un nodo ganador.

RF10 Resolver manualmente los conflictos de creación de la estructura para un nodo no ganador.

RF11 Resolver manualmente los conflictos de modificación de la estructura para un nodo ganador.

RF12 Resolver manualmente los conflictos de modificación de la estructura para un nodo no ganador

RF13 Resolver manualmente los conflictos de no existencia de la estructura para un nodo no ganador.

RF14 Resolver los conflictos desconocidos.

RF15 Monitorizar los conflictos de estructura.

2.3.2 Requisitos no funcionales

Los requisitos no funcionales definen las restricciones del sistema, son propiedades que el sistema debe poseer. Representan las características del producto.(45)

Los requisitos no funcionales del módulo a desarrollar se representan en la siguiente tabla:

Tabla 3 Requisitos no funcionales

No.	Atributo de calidad	Sub-Atributo	Objetivo
RnF1	Usabilidad	Operabilidad	Capacidad que presenta el producto que le permite al usuario operarlo y controlarlo con facilidad.
		Protección contra errores de usuarios	Capacidad que presenta el producto para proteger a los usuarios de cometer errores cuando realiza alguna acción en el sistema.
RnF2	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.
RnF3	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 que presenta el producto de permitir modificaciones 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.
RnF4	Portabilidad	Adaptabilidad	Capacidad que presenta el producto que le permite ser adaptado de forma efectiva y eficiente a diferentes entornos determinados de hardware, software, operacionales o de uso.
RnF5	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.
RnF6	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 del equipo de desarrollo

Para una mejor comprensión de los requisitos no funcionales y sus atributos de calidad ver Anexo del 2 al 11. Hay requisitos de calidad que con sus respectivos atributos no han sido ejemplificados porque están incluidos en el sistema actual y su interpretación se describe en el documento Especificación de requisitos referente al expediente de proyecto del Replicador de datos REKO.

2.4 Propuesta de solución

Para la solución del problema planteado se extenderán las funcionalidades del componente Consola Web de Administración con el objetivo de gestionar las configuraciones y monitorización de los conflictos de estructura. En el módulo Configuración General se realizará una configuración previa a la resolución de los conflictos de estructura, donde se define cómo debe comportarse el nodo ante la información que recibe desde un nodo remoto. Por tanto, se deberá extender el funcionamiento del módulo Configuración General para brindarle al usuario

la posibilidad de establecer las configuraciones de los conflictos de estructura, donde se identifica el tipo de nodo y variante de solución. Para la incorporación de las interfaces de usuario en la resolución de conflictos de forma manual se deberá extender el módulo de Conflictos. Para monitorizar todos los eventos referentes a la captura y solución de los conflictos, se hace necesario extender el módulo Monitor de Réplica.

Si al aplicar un cambio DDL se genera un error en la BD se procederá la **captura** del conflicto. En el componente Aplicador serán extendidas las funcionalidades que permitan capturar y clasificar el conflicto una vez que el cambio arribe a su destino y se intente aplicar, según el tipo de acción que se ejecute, el error generado por el gestor mediante el *SQL State*[35] y la configuración definida para la resolución de conflictos.

Una vez capturado el conflicto, se clasificara en alguno de los siguientes tipos:

- Conflicto de **creación de estructuras**: ocurre cuando se intenta crear una estructura que ya existe en la BD o en la relación actual.
- Conflicto de **modificación de estructuras**: sucede cuando se quiere modificar una estructura y la definición actual de la estructura no permite la ejecución del cambio, o porque otras relaciones no permiten su ejecución producto a su propia definición.
- Conflicto de **eliminación de estructuras**: acontece cuando se desea eliminar una estructura que otras relaciones emplean en su definición.

Cuando el nodo local entre en conflicto serán **monitorizadas** las acciones de las instancias involucradas. Para monitorizar el proceso de resolución de conflictos generados durante la réplica de estructura se hace necesario extender las funcionalidades del módulo Monitor de Réplica, donde mostrará todos los eventos referentes a las acciones que se encuentren en conflicto.

Una vez capturado el conflicto debe ser **resuelto**, si el nodo local no está en conflicto y no tiene grupos pendientes por ser aplicados se pasará a resolver, de lo contrario será persistido en una BD embebida para esperar a que pueda ser aplicado. Otro de los componentes que se extenderán es el Distribuidor, en el cual se realizará la resolución de conflictos generados y las notificaciones referentes a los cambios de estructura.

[35] Error que se muestra cuando se ejecuta una acción incorrecta en el gestor de BD, en el panel de salida en la pestaña mensaje.

Si la resolución del conflicto es automática se resolverá según la configuración que se haya definido en el módulo Configuración General. Si la configuración definida es manual, en el momento que se genere el conflicto el usuario podrá decidir que variante aplicar para dar solución al mismo.

El conflicto será resuelto según la variante definida por el usuario a través de las configuraciones. Para definir las variantes de solución al conflicto capturado, se consideraron las siguientes condiciones.

- Si se define el nodo remoto como *Ganador* entonces la información recibida que entra en conflicto tiene prioridad y prevalece sobre la existente en el nodo local.
- Si se define el nodo remoto como *No Ganador* entonces la información recibida que entra en conflicto no tiene prioridad sobre la existente en el nodo local.
- En cualquier variante de solución que sea aplicada, debe mantenerse la consistencia entre las BD.

A continuación se muestran las **variantes** definidas para los diferentes tipos de conflictos de estructura:

Resolución de conflictos de **creación de estructuras** muestra la manera de resolver los conflictos de creación de las estructuras *(Automático* o *Manual)*.

Variantes para un nodo ganador: la estructura local no se modifica y tiene prioridad sobre la recibida.

- *Ignorar cambio:* ignora los cambios del proceso de réplica de estructura que llegan. Para mantener la consistencia de las BD se envía una modificación de la estructura al nodo remoto.
- *Generar nombre de estructura:* genera un nuevo nombre para la estructura que llega. Se envía una modificación al nombre a la estructura del nodo remoto para mantener la consistencia de las BD.

Variantes para un nodo no ganador: la estructura local no tiene prioridad sobre la recibida y puede ser modificada.

- *Aplicar estructura:* elimina la estructura actual y crea la recibida.
- *Modificar estructura:* genera un nuevo nombre para la estructura actual y crea la recibida.

Resolución de conflictos de **modificación de estructuras** muestra la manera de resolver los conflictos de modificación de las estructuras *(Automático* o *Manual)*.

Variantes para un nodo ganador: la estructura local no se modifica y tiene prioridad sobre la recibida.

- *Ignorar cambio:* ignora los cambios del proceso de réplica de estructura que llegan. Para mantener la consistencia de las BD se envía una modificación a la estructura del nodo remoto.
- *Generar nombre de estructura:* genera un nuevo nombre para la estructura que llega. Se envía una modificación al nombre de la estructura del nodo remoto para mantener la consistencia de las BD. Esta variante no se aplica para las acciones referentes a las columnas (ejemplo de estas acciones: adicionar una columna, cambiarle el nombre, el tipo de dato y que no acepte valores nulos), debido a que al generar un nuevo nombre para la columna y enviar la modificación al nodo remoto no se tendría la misma estructura en las BD. Así como para las acciones referentes a la llave primaria ya que las tablas no aceptan múltiples llaves primarias.

Variantes para un nodo no ganador: la estructura local no tiene prioridad sobre la recibida y puede ser modificada.

- *Aplicar estructura:* elimina la estructura actual y crea la recibida.
- *Modificar estructura:* genera un nuevo nombre para la estructura actual y crea la recibida. Esta variante no se aplica para las acciones referentes a las columnas (ejemplo de estas acciones: adicionar una columna, cambiarle el nombre, el tipo de dato y que no acepte valores nulos), debido a que al generar un nuevo nombre para la columna y enviar la modificación al nodo remoto no se tendría la misma estructura en la BD. Así como para las acciones referentes a la llave primaria ya que las tablas no aceptan múltiples llaves primarias.

Resolución de conflictos de **eliminación de estructuras** muestra la manera de resolver los conflictos de eliminación de las estructuras *(Automático* o *Manual)*.
Variantes para un nodo ganador: la estructura local no se modifica y tiene prioridad sobre la recibida.

- *Ignorar cambio:* ignora los cambios del proceso de réplica de estructura que llegan.

Variantes para un nodo no ganador: la estructura local no tiene prioridad sobre la recibida y puede ser modificada.

- *Ignorar cambio:* ignora los cambios del proceso de réplica de estructura que llegan. No se elimina la estructura actual porque otras instancias replican sobre ella.

- *Eliminar estructura:* elimina la estructura actual.

Si la configuración realizada es manual se mostrará una interfaz que considera si el nodo local es ganador o no ganador y el tipo de conflicto generado, brindará variantes que permitan al usuario decidir el proceder del software para dar solución al conflicto, donde se respeta la prioridad de la información. Si existe una configuración manual para un conflicto de eliminación de la estructura y el nodo que recibe la información es ganador el sistema automáticamente ignorará el cambio.

Cuando el nodo local resuelva el conflicto serán **monitorizadas** las acciones de las instancias involucradas con el conflicto. Se mantendrá informado al usuario local y remoto del proceso realizado para resolver el conflicto.

2.5 Historias de usuario

La HU es una representación de un requisito de software escrito en una o dos frases con la utilización del lenguaje común del usuario. Se utilizan en las metodologías de desarrollo ágiles para la especificación de requisitos. Son una forma rápida de administrar los requisitos sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Las historias de usuario permiten responder rápidamente a los requisitos cambiantes.(46)

2.5.1 Descripción de las HU

Para comprender las HU del módulo para la resolución de conflictos se muestran dos ejemplos en las siguientes tablas. Las restantes HU referentes al módulo están representadas a partir del Anexo 12 hasta el 24.

Tabla 4 HU 1 Guardar configuración para la resolución de conflictos de estructura

Historias de Usuario	
Número: HU 1	**Nombre del requisito:** Guardar la configuración para la resolución de conflictos de estructura.
Programador: Osvaldo Cuba López	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 7 días
Riesgo en Desarrollo: Plan de riesgos	**Tiempo Real:** 56 horas
Descripción: para realizar la configuración general se crea la sección “Conflictos de estructura” en el módulo “Configuración General”, el sistema debe permitir al usuario	

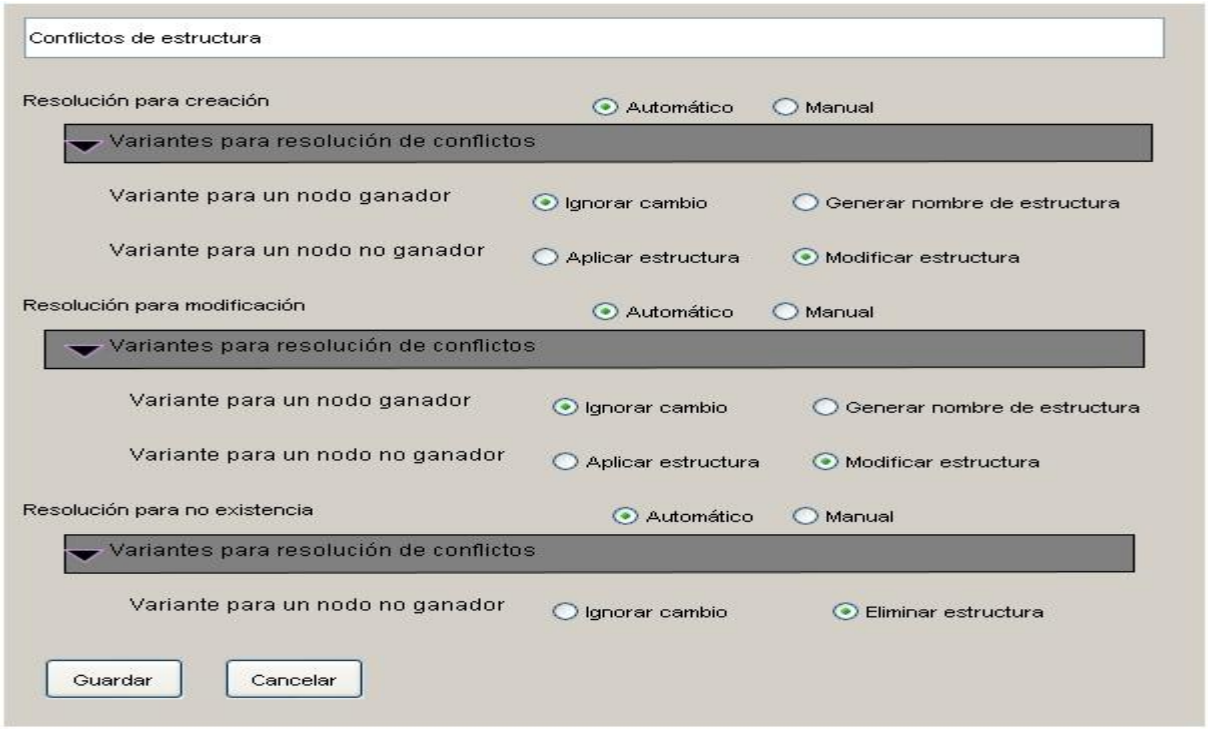
determinar los parámetros de configuración (como se muestra en el prototipo no funcional) para la resolución de conflictos de estructura. Cuando el usuario establezca los parámetros editables de la configuración (Conflictos de estructura) y de clic en la opción “Guardar”, el sistema salva la configuración. El sistema debe brindar al usuario la posibilidad de definir si desea que la configuración sea manual o automática con las variantes de solución que se defina para la resolución de conflictos de estructura.

Si el usuario selecciona la opción “Manual”, el sistema selecciona dicha opción para el tipo de acción que se desee resolver (conflictos de creación de la estructura, de modificación de la estructura o de no existencia de la estructura).

Si el usuario selecciona la opción “Automática”, el sistema selecciona dicha opción y muestra las variantes para la resolución de conflictos para un nodo ganador o para un nodo no ganador para el tipo de acción que se desee resolver para esa opción (conflictos de creación de la estructura, de modificación de la estructura o de no existencia de la estructura).

Observaciones: la configuración se realiza sobre una configuración por defecto que se tiene en el sistema.

Prototipo de interfaz:



Fuente: elaboración propia del equipo de desarrollo

Tabla 5 HU 3 Resolver automáticamente los conflictos de creación de la estructura para un nodo ganador

Historias de Usuario	
Número: HU 3	**Nombre del requisito:** Resolver automáticamente los conflictos de creación de la estructura para un nodo ganador.
Programador: Liset Martínez Almaguer	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 7 días
Riesgo en Desarrollo: Plan de riesgos	**Tiempo Real:** 56 horas
Descripción: la clase administradora de conflictos de estructura recibe un conflicto de creación de la estructura. Una vez que se genere un conflicto el sistema debe monitorizar la acción en conflicto y enviar una notificación para que el nodo remoto monitorice que el nodo local entró en conflicto. Si en la configuración está activada la solución automática para este tipo de conflicto, procede al flujo automático. A partir de la configuración del módulo de Conflictos si el nodo local es ganador, y está seleccionada la variante "Ignorar cambio", para dar solución al conflicto se elimina la acción que está en conflicto del grupo de estructura replicable y aplica el grupo de estructura replicable sin conflicto sobre la base de datos, luego de este proceso se envía una notificación al nodo remoto con la estructura que se tiene en el nodo local para que actualice su estructura y reanuda el proceso de réplica. Si la variante que está seleccionada es "Generar nombre de estructura" la acción que se recibió y entró en conflicto se le cambia el nombre y aplica el grupo de estructura replicable sin conflicto sobre la base de datos, envía una notificación al nodo remoto para que actualice la estructura con el nombre nuevo y se reanuda el proceso de réplica. Luego de resuelto el conflicto se monitoriza dicho proceso de solución y envía una notificación al nodo remoto para que monitorice la resolución del conflicto en el nodo local.	
Observaciones: NA	
Prototipo de interfaz: NA	

Fuente: elaboración propia del equipo de desarrollo

2.6 Arquitectura del Replicador de datos REKO

La arquitectura de software es la estructura u organización de los componentes del programa (módulos), la manera en que estos componentes interactúan, y la estructura de datos que utilizan los componentes.(27)

La arquitectura brinda una visión global del sistema. Permite entenderlo, organizar su desarrollo, plantear la reutilización del software y hacerlo evolucionar. Exige diseñar muy cuidadosamente la arquitectura bajo la cual funciona el sistema, ya que las decisiones que se tomen tendrán influencia a lo largo de todo el ciclo de vida de la aplicación. Por su definición es necesario seleccionar y combinar patrones.(47)

Los patrones de diseño pueden usarse durante el diseño del software. Una vez desarrollado el modelo de análisis, el diseñador puede examinar una representación detallada del problema que debe resolver y las restricciones que impone el problema. La descripción del problema se examina en varios grados de abstracción para determinar si es flexible para uno o más de los siguientes tipos de patrones de diseño.

- Patrones arquitectónicos: estos patrones definen la estructura general del software, indican las relaciones entre los subsistemas y los componentes del software y definen las reglas para especificar las relaciones entre los elementos (clases, paquetes, componentes, subsistemas) de la arquitectura.
- Patrones de diseño: estos patrones se aplican a un elemento específico del diseño como un agregado de componentes para resolver algún problema de diseño, relaciones entre los componentes o los mecanismos para efectuar la comunicación de componente a componente.
- Idiomas: a veces llamados *"patrones de código"* estos patrones específicos del lenguaje, por lo general implementan un elemento algorítmico o un componente, un protocolo de interfaz específico o un mecanismo de comunicación entre los componentes.(27)

2.6.1 Estilo y patrones arquitectónicos

El estilo arquitectónico **Llamada y retorno** es el utilizado 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 de gran escala. Miembros de la familia son las arquitecturas en capas, el modelo-vista-controlador, los sistemas orientados a objeto y los sistemas basados en componentes. (47)

Arquitectura basada en componentes

En la especificación UML se define que: "…un componente es una unidad modular con interfaces bien definidas, que es reemplazable dentro del contexto". (44)

La arquitectura basada en componentes se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas. A su vez los componentes de software son unidades reutilizables que pueden relacionarse con otros módulos de software por medio de interfaces bien definidas que contienen métodos, eventos y propiedades, y pueden estar escritos en lenguaje funcional, procedural u orientado a objetos. (47)

Las definiciones expuestas se aplican para describir en términos generales la arquitectura del Replicador de datos REKO, 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 módulo propuesto. En el paquete **conflic.structure**, correspondiente al componente Distribuidor, se le realizarán modificaciones, pues hasta el momento su funcionamiento se basa en determinar el destino de cada cambio realizado en la BD, los envía y se responsabiliza de su llegada, ahora se extenderá su funcionamiento para resolver conflictos de estructura. Con este mismo fin serán modificados los paquetes del Capturador, Aplicador y Administrador.

Arquitectura en capas

Independientemente de lo antes expuesto, 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.(47)

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.(47)

Por lo antes expuesto se muestra en la figura 3 los principales componentes del módulo para la resolución de conflictos en el proceso de réplica de estructura del Replicador de datos REKO.

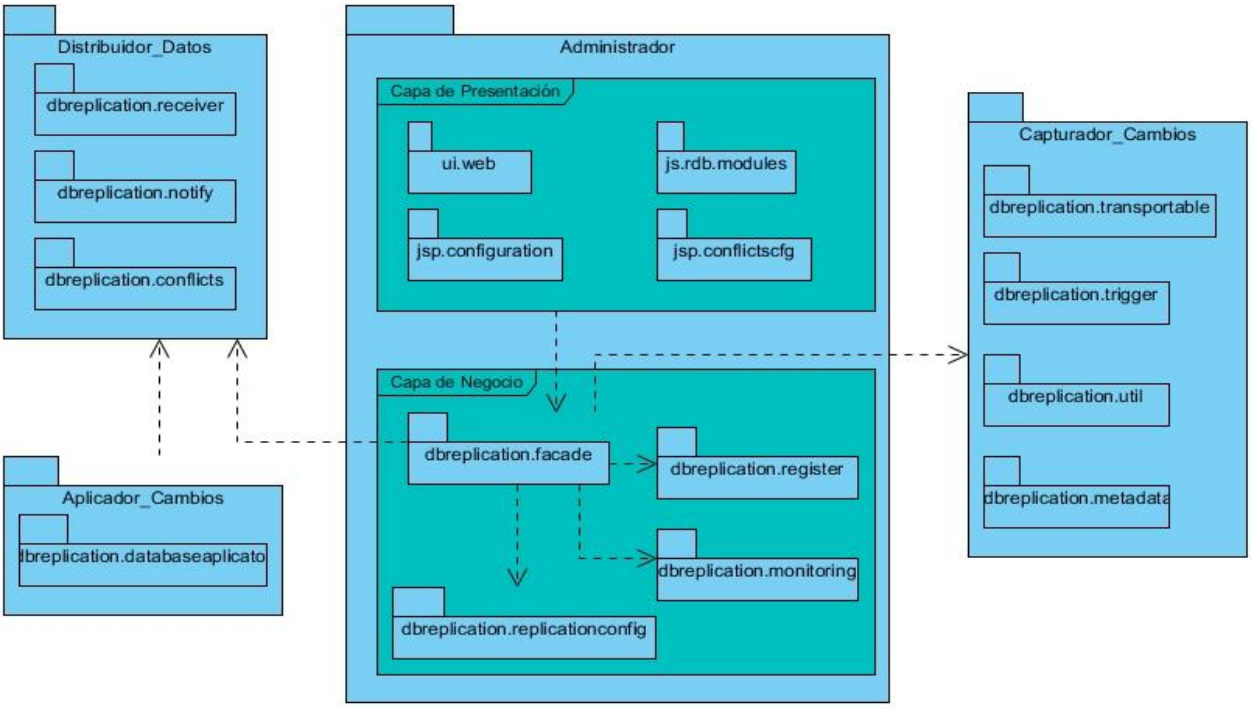


Figura 3 Vista de los principales componentes del módulo para la resolución de conflictos

Fuente: elaboración propia del equipo de desarrollo

2.6.2 Patrones de diseño

Un patrón de diseño constituye una solución estándar para un problema común de programación en el desarrollo del software. Además es una técnica muy eficaz para flexibilizar el código haciéndolo satisfacer ciertos criterios, así como permite una manera más práctica de describir ciertos aspectos de la organización de un programa.(48)

Para el desarrollo del módulo se usaron los patrones de asignación de responsabilidades, conocidos como patrones **GRASP**[36] los cuales describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.(44)

- **Experto:** en el desarrollo del módulo para la resolución de conflictos generados en el proceso de réplica de estructura se tuvo en cuenta el patrón experto ya que cada clase cuenta con la información necesaria para cumplir con la responsabilidad que se le asigna. Por ejemplo la clase StructureConflictManager es la encargada de manejar toda la información referente a la resolución de conflictos. La clase NotifyManager es la encargada de realizar las notificaciones relacionadas a la resolución de conflictos, otra de

[36] Patrones Generales de Software para Asignar Responsabilidades, del inglés *General Responsibility Assignment Software Patterns.*

las clases es la MonitoringManager que es la que se encarga de realizar la monitorización de los eventos referentes a la resolución de conflictos.

- **Controlador**: un controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Define además el método de su operación. En el módulo desarrollado se utilizan distintas clases controladoras como la StructureConflictManager que es la encargada de manejar todos los eventos asociados a la resolución de los conflictos de estructura.
- **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.
- **Bajo acoplamiento:** las clases se comunican solamente con las clases necesarias para desarrollar cada flujo de evento. Por ejemplo, la clase StructureConflictManager se comunica con la menor cantidad de clases posibles para el flujo de evento de la resolución de conflictos de estructura y presenta así un bajo acoplamiento.
- **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. El uso de este patrón se manifiesta en la clase StructureConflictException donde se tienen presente los tipos de excepciones relacionadas con los conflictos detectados (ver Anexo 25).(44)

Además se utilizaron los patrones **GOF**[37] los cuales se clasifican en dependencia del propósito para los que hayan sido definidos: creación, estructurales y de comportamiento.(49)

- **Fachada**: provee de una interfaz unificada simple, para 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 cambian o se sustituyen por otras, solamente se tiene que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones cliente. Ejemplo de lo anterior se evidencia en la propuesta de solución mediante el componente facade que aprovisiona estas ventajas, la clase ReplicationFacade como interfaz, así como ReplicationFacadeImpl ofrece un punto de acceso a la configuración del módulo de conflictos.(49)

[37] Banda de los Cuatro, del inglés *Gang-Of-Four.*

Otro de los patrones utilizados para la solución propuesta es el patrón **DAO**[38], se dividen más las responsabilidades en la aplicación de tal forma que tendremos clases que se encargarán de la lógica de negocio y otras clases de la responsabilidad de persistencia. Fuera de las clases DAO no debe haber ningún tipo de código que acceda al repositorio de datos. Los Objetos de Acceso a Datos pueden usarse en Java para aislar a una aplicación de la tecnología de persistencia Java subyacente, la cual es JDBC (ver Anexo 26).(50)

2.7 Modelo de diseño

Un modelo de diseño es un modelo de objetos que se centra en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar.(51)

2.7.1 Diagramas de paquetes

Un diagrama de paquetes muestra cómo un sistema está dividido en agrupaciones lógicas y las dependencias entre esas agrupaciones. Los paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre los paquetes. Cada paquete puede asignarse a un individuo o a un equipo, y las dependencias entre ellos pueden indicar el orden de desarrollo requerido.(52)

En la siguiente figura se muestra uno de los diagramas de paquetes del módulo propuesto, los restantes se encuentran en los anexos referentes a la investigación (ver Anexo del 27 al 30).

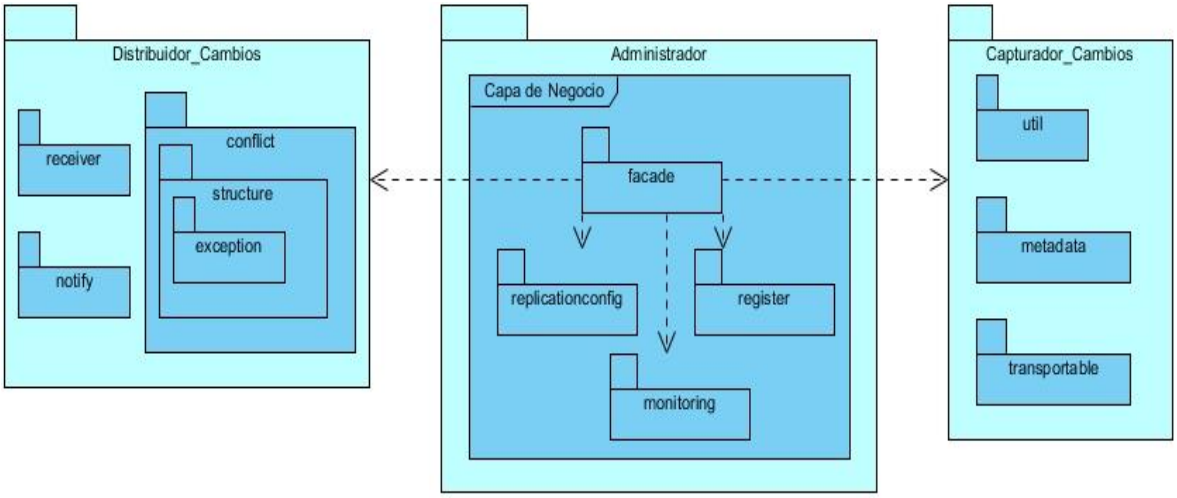


Figura 4 Diagrama de paquetes de los RF 3, RF 4, RF 5, RF 6, RF 7, RF 8 y RF 14

Fuente: elaboración propia del equipo de desarrollo

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

2.7.2 Diagramas de clases del 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.(53)

En la siguiente figura se muestra el diagrama de clases del diseño referente al diagrama de paquetes de los RF 3, RF 4, RF 5, RF 6, RF 7, RF 8 y RF14, los restantes diagramas del diseño están reflejados en los Anexo del 31 al 34.

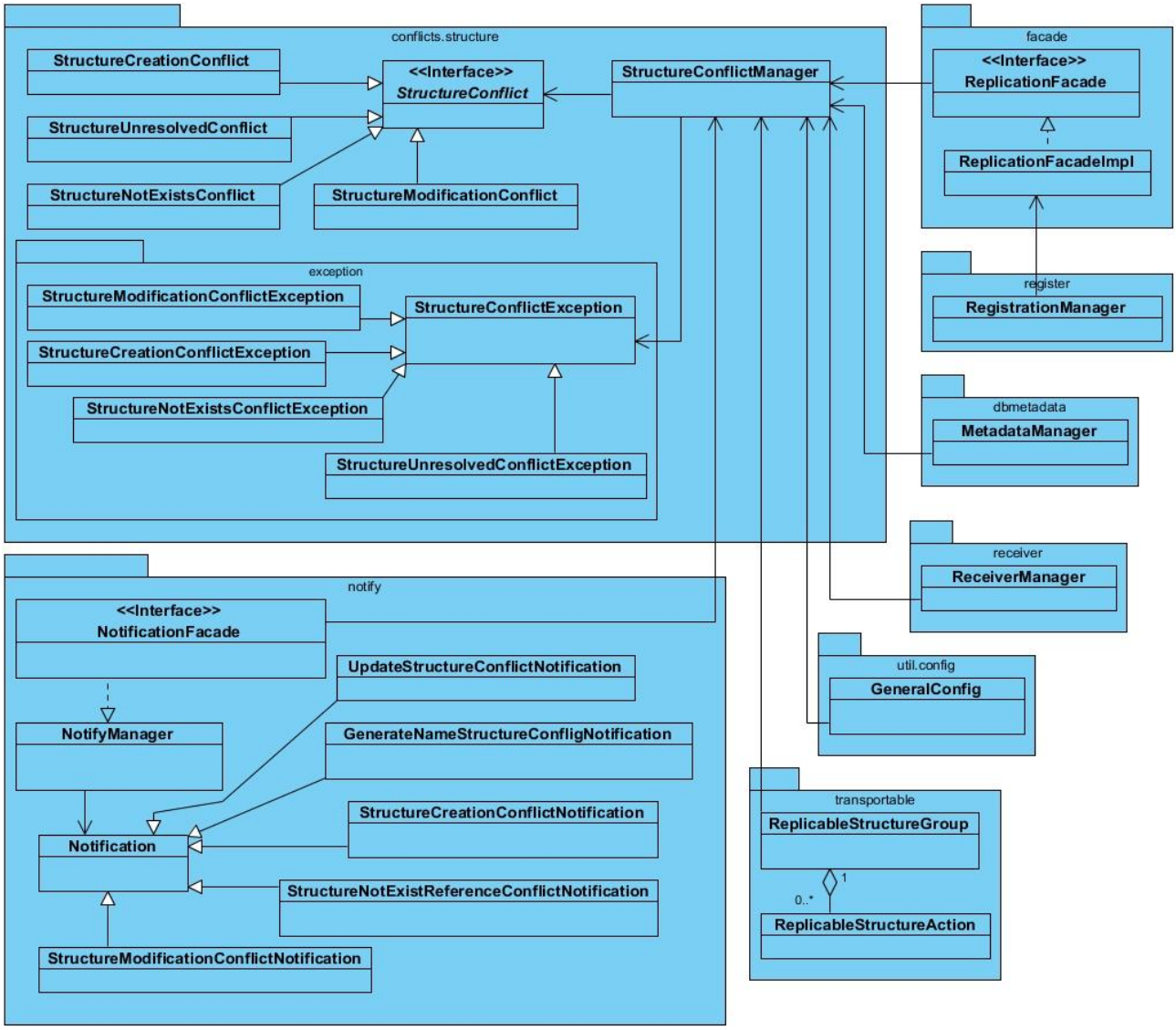


Figura 5 Diagrama de clases del Diagrama de paquetes del RF 3, RF 4, RF 5, RF 6, RF 7, RF 8 y RF 14
Fuente: elaboración propia del equipo de desarrollo

2.7.3 Descripción de las clases del diseño

A continuación se realizan las descripciones de las clases más importantes. (Ver tablas 6 y 7)

Tabla 6 Descripción textual de la clase StructureConflictManager

Descripción de la clase StructureConflictManager	
Nombre: StructureConflictManager.	
Tipo de clase: Controladora.	
Atributos	**Tipos**
conflictReplicableStructureGroupDao	private
metadataManager	private
receiverManager	private
registrationManager	private
notificationFacade	private
monitoringManager	private
berkeleyConfigurationDaoImpl	private
Responsabilidades	
Nombre:	persistWaitingRSGbyApply(replicableStructureGroup : cu.uci.dbreplication.transportable.ReplicableStructureGroup) : void
Descripción:	Persiste en Berkeley el grupo replicable en conflicto.
Nombre:	solveStructureConflict(replicableStructureGroup : cu.uci.dbreplication.transportable.ReplicableStructureGroup, structureConflictDatabaseException : cu.uci.dbreplication.conflicts.structure.exception.StructureConflictException,c: int) : void
Descripción:	A partir de la configuración del módulo y el tipo de conflicto elige un tratamiento específico para cada conflicto.
Nombre:	solveManualStructureConflict(replicableStructureActionConflict: cu.uci.dbreplication.conflicts.structure. ReplicableStructureActionConflict, c:int): void
Descripción:	A partir de la variante seleccionada por el usuario y el tipo de acción que se encuentre en conflicto, se resuelve el conflicto generado.
Nombre:	existReplicableStructureGroupInWaiting() : boolean
Descripción:	Pregunta si existen grupos de estructura en espera de ser aplicados.

Nombre:	getFirstReplicableStructureGroupInWaiting() : cu.uci.dbreplication.transportable.ReplicableStructureGroup
Descripción:	Obtiene el primer grupo de estructura replicable en espera de ser aplicado.
Nombre:	deleteReplicableStructureGroupInWaiting(replicableStructureGroup : cu.uci.dbreplication.transportable.ReplicableStructureGroup) : boolean
Descripción:	Elimina el grupo de estructura replicable en espera.
Nombre:	getAlterVariantLoserNode():String
Descripción:	Retorna la variante que está seleccionada para un nodo no ganador en la acción donde se modifique la relación.
Nombre:	getAlterVariantWinnerNode():String
Descripción:	Retorna la variante que está seleccionada para un nodo ganador en la acción donde se modifique la relación.
Nombre:	getCreateVariantLoserNode():String
Descripción:	Retorna la variante que está seleccionada para un nodo no ganador en la acción de creación de la relación.
Nombre:	getCreateVariantWinnerNode():String
Descripción:	Retorna la variante que está seleccionada para un nodo ganador en la acción de creación de la relación.
Nombre:	getDropVariantLoserNode() : String
Descripción:	Retorna la variante que está seleccionada para un nodo no ganador en la acción donde no exista la relación.
Nombre:	configurationTableAndSchema(replicableStructureActionConflict : cu.uci.dbreplication.transportable.ReplicableStructureAction) : String
Descripción:	Retorna la estructura eliminada de la configuración de réplica.
Nombre:	configurationNode(replicableStructureActionConflict : cu.uci.dbreplication.transportable.ReplicableStructureAction) : String
Descripción:	Retorna el nodo que tenga una configuración de réplica con el esquema o la tabla que está en conflicto.
Nombre:	generateNameStructureConflig(oldName : String, actionStructureResult : cu.uci.dbreplication.transportable.ReplicableStructureAction) : String
Descripción:	Retorna la sentencia para cambiar el nombre a un determinado

	esquema o tabla que se encuentre en conflicto.
Nombre:	configurationRenameStructure(replicableStructureAction : cu.uci.dbreplication.transportable.ReplicableStructureAction, oldName : String, newName : String) : void
Descripción:	Cambia el nombre en la configuración de réplica de un esquema o de una tabla que esté en conflicto.
Nombre:	sqlActionConflict(replicableStructureAction : cu.uci.dbreplication.transportable.ReplicableStructureAction) : String
Descripción:	Retorna el sql de la acción que este en conflicto.
Nombre:	actionTypeConflicStructure(structureConflictDatabaseException : cu.uci.dbreplication.conflicts.structure.exception.StructureConflictException) : int
Descripción:	Retorna un número según el tipo de acción que esté en conflicto.
Nombre:	deleteConfigTableAndSchema(replicableStructureGroup : cu.uci.dbreplication.transportable.ReplicableStructureGroup, replicableStructureAction : cu.uci.dbreplication.transportable.ReplicableStructureAction) : void
Descripción:	Se elimina la tabla o el esquema que tenga configuración de réplica (en el nodo local).

Fuente: elaboración propia del equipo de desarrollo

Tabla 7 Descripción textual de la clase StructureConflict

Descripción de la clase StructureConflict	
Nombre: StructureConflict	
Tipo de clase: Interface	
Atributos	**Tipos**
replicableStructureActionConflict	private
Responsabilidades	
Nombre:	applyChange(replicableStructureGroup: cu.uci.dbreplication.transportable.ReplicableStructureGroup, replicableStructureActionConflict: cu.uci.dbreplication.transportable.ReplicableStructureAction, metadataManager:cu.uci.dbreplication.dbmetadata.MetadataManager, monitoringManager:cu.uci.dbreplication.monitoring.MonitoringManager):

	cu.uci.dbreplication.transportable.ReplicableStructureAction
Descripción:	Devuelve la acción que es resuelta, la cual estaba anteriormente en conflicto.
Nombre:	ignoreChange(replicableStructureGroup: cu.uci.dbreplication.transportable.ReplicableStructureGroup, structureConflictDatabaseException: cu.uci.dbreplication.conflicts.structure.exception.StructureConflictException,receiverManager:cu.uci.dbreplication.receiver.ReceiverManager) : void
Descripción:	Elimina la acción del grupo de estructura replicable que está en conflicto.
Nombre:	createStructure(replicableStructureAction: cu.uci.dbreplication.transportable.ReplicableStructureAction, metadataManager: cu.uci.dbreplication.dbmetadata.MetadataManager) : void
Descripción:	Crea las estructuras que no se encuentren en la BD y son necesarias para resolver el conflicto.
Nombre:	ignoreUpdate(replicableStructureActionConflict: cu.uci.dbreplication.transportable.ReplicableStructureAction, metadataManager cu.uci.dbreplication.dbmetadata.MetadataManager) : String
Descripción:	Retorna la sentencia de la estructura que se tiene en la BD de la acción que entró en conflicto.
Nombre:	generateName(replicableStructureGroup: cu.uci.dbreplication.transportable.ReplicableStructureGroup, replicableStructureActionConflict: cu.uci.dbreplication.transportable.ReplicableStructureAction, metadataManager: cu.uci.dbreplication.dbmetadata.MetadataManager): cu.uci.dbreplication.transportable.ReplicableStructureAction
Descripción:	Retorna una acción con un nuevo nombre.
Nombre:	updateStructure(replicableStructureActionConflict: cu.uci.dbreplication.transportable.ReplicableStructureAction, metadataManager:

	cu.uci.dbreplication.dbmetadata.MetadataManager,nodeRemoto: String) : String
Descripción:	Retorna la sentencia de cambiarle el nombre a la estructura de la BD que se encuentra en conflicto.

Fuente: elaboración propia del equipo de desarrollo

Se realizó la descripción de otras clases donde se recurrió a métodos ya implementados y en otros casos fue necesaria la implementación de nuevas operaciones. (Ver Anexo del 35 al 40)

2.8 Modelo de despliegue

Establece una correspondencia entre la arquitectura de software y la arquitectura de hardware del sistema. 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 los nodos procesadores, la distribución de los procesos y los componentes.(44)

El modelo de despliegue de la propuesta de solución se muestra en la siguiente figura:

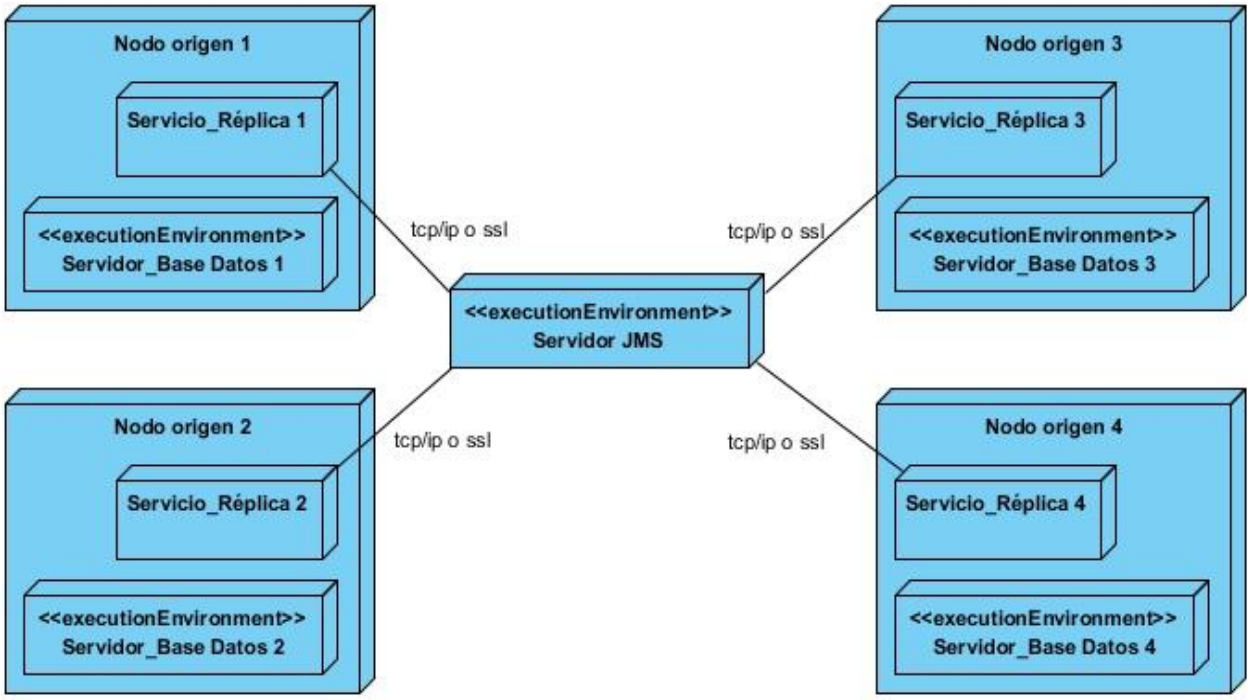


Figura 6 Diagrama de despliegue

Fuente: elaboración propia del equipo de desarrollo

2.9 Consideraciones del capítulo

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

- El análisis del dominio de aplicación junto a las necesidades del cliente, permitió identificar los requisitos funcionales y no funcionales, para garantizar el funcionamiento del módulo para la resolución de conflictos en el proceso de réplica de estructura.
- La caracterización del proceso de réplica de estructura refleja insuficiencia en el tratamiento de conflictos por lo que se extenderán las funcionalidades de los componentes Distribuidor, Aplicador, Administrador y Capturador.

CAPÍTULO 3. IMPLEMENTACIÓN Y VALIDACIÓN

En el presente capítulo se ejemplifica 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 es comprendido por un conjunto de componentes y subsistemas que representan la composición física de la implementación del sistema. Entre estos componentes se encuentran datos, archivos, ejecutables, código fuente y directorios. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos.(54)

3.1.1 Diagramas de componentes

Los diagramas de componentes muestran los componentes de software (código fuente, binario y ejecutable, bibliotecas dinámicas, páginas web, interfaces y un conjunto de relaciones de dependencia, generalización, asociación y realización) y las relaciones lógicas entre ellos en un sistema. Uno de los usos principales es identificar qué componentes pueden ser compartidos entre las partes de un sistema o entre distintos sistemas, al modelar la vista estática del sistema.(44)

A continuación, para comprender la composición física de la implementación del módulo, en la siguiente figura (ver Figura 9) se muestra un diagrama de componentes agrupados por los sub-sistemas implicados en la solución. Los restantes diagramas de componentes relacionados con la solución se pueden ver en los anexos del 41 al 43.

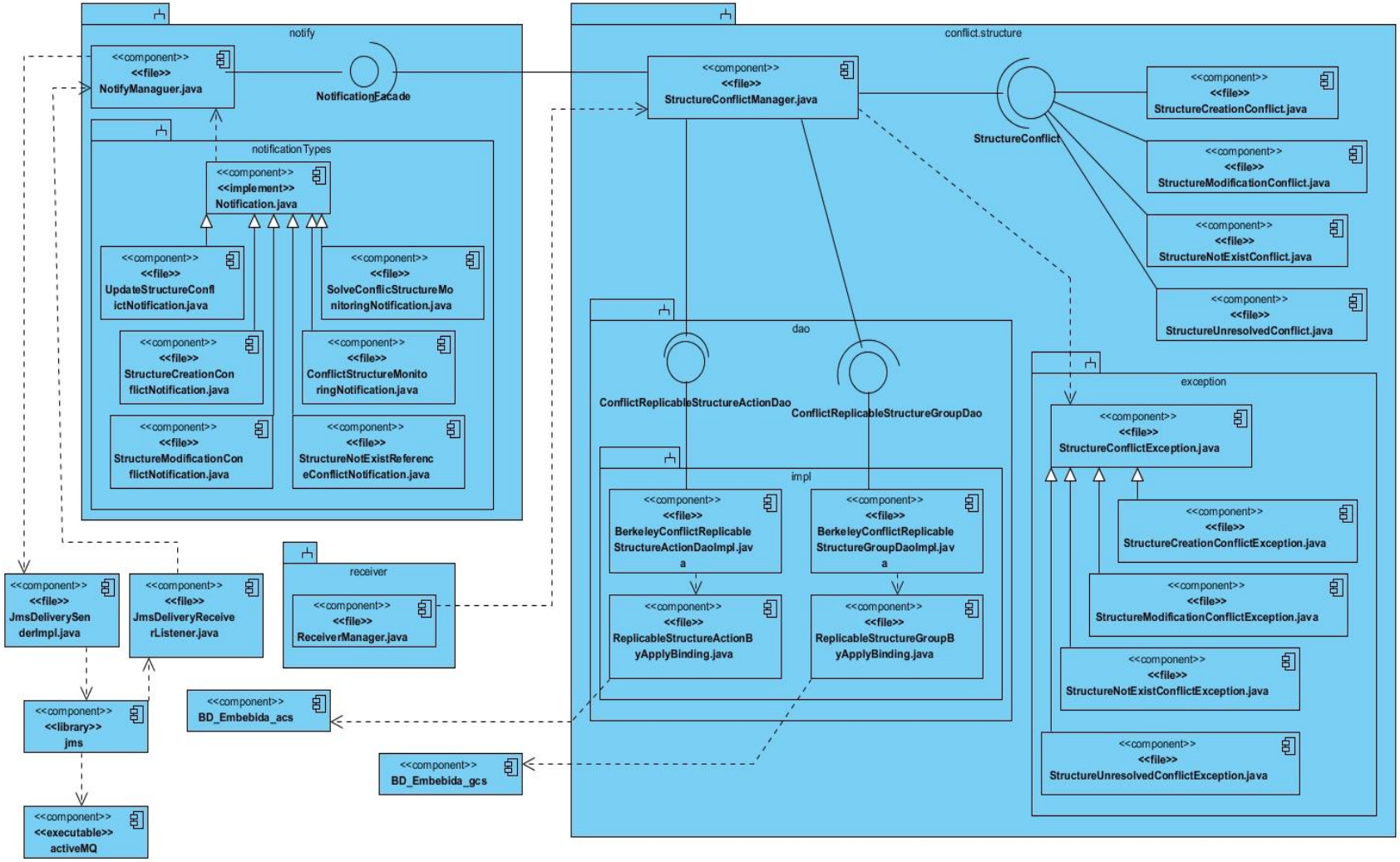


Figura 7 Diagrama de componentes para el subsistema Distribuidor de Datos

Fuente: elaboración propia del equipo de desarrollo

3.2 Código fuente

El código fuente es un conjunto de líneas que conforman un bloque de texto, escrito según las reglas sintácticas de algún lenguaje de programación destinado a ser legible por humanos. Es un programa en su forma original, tal y como fue escrito por el programador, no es ejecutable directamente por el computador, debe convertirse en lenguaje de máquina mediante compiladores, ensambladores o intérpretes.(55)

El siguiente método perteneciente a la clase StructureNotExistsConflict (ver Tabla 8), se utiliza para resolver el conflicto que se genere en caso que se desee eliminar un objeto en la BD que tenga una dependencia con otro, a partir de la configuración establecida.

Tabla 8 Código fuente del método que se encarga de resolver el conflicto de no existencia de la estructura

StructureNotExistsConflict

```
@Override
      public ReplicableStructureAction
applyChange(ReplicableStructureAction
replicableStructureActionConflict,MetadataManager metadataManager)
      {
            ReplicableStructureAction replicableStructureAction = new
ReplicableStructureAction();
 try
{
   HibernateDialect dialect = DialectFactory.getDialect(metadataManager);
   Connection con = null;
   Statement stt = null;
   String dropTable;
   String dropColum;
   String dropConstraint;
   con = metadataManager.getConnection();
   stt = con.createStatement();
/*en el caso que se desee eliminar un esquema que tangan tablas*/
if(replicableStructureActionConflict.getActionType().equals(ActionType.drop
_schema))
      {
            dropSchema = DialectUtils.sentenceDropSchemaReference(dialect,
                  replicableStructureActionConflict.getSchemaName());
            stt.executeUpdate(dropSchema);
            replicableStructureAction = null;
      }
/*en el caso que se desee eliminar una tabla que otras tablas dependan de
ella */
      if(replicableStructureActionConflict.getActionType().equals(ActionTyp
e.drop_table))
      {
           dropTable = DialectUtils.sentenceDropTableReference(dialect,
           replicableStructureActionConflict.getTableName());
           stt.executeUpdate(dropTable);
           replicableStructureAction = null;
      }
```

```
/*en el caso que se desee eliminar una columna que otras tablas dependan de
ella*/
      else
if(replicableStructureActionConflict.getActionType().equals(ActionType
       .drop_colum))
      {
           dropColum = DialectUtils.sentenceDropColumReference(dialect,
                       replicableStructureActionConflict.getTableName(),
                 replicableStructureActionConflict.getColumName());
           stt.executeUpdate(dropColum);
           replicableStructureAction = null;
      }
/*en el caso que se desee eliminar un constraint que otras tablas dependan
de él*/
      else if(replicableStructureActionConflict.getActionType().
equals(ActionType.alter_add_primaryKey))
      {
           dropConstraint = DialectUtils.sentenceDropPkReference(dialect,
                replicableStructureActionConflict.getTableName(),
replicableStructureActionConflict.getOtherEspecification());
           stt.executeUpdate(dropConstraint);
           replicableStructureAction = null;
      }
      }
     catch (Exception e)
     {
      e.printStackTrace();
     }
        return replicableStructureAction;
}
```

Fuente: elaboración propia del equipo de desarrollo

3.3 Pruebas del software

Las pruebas de software constituyen una fase del proceso de desarrollo de un software centrada en la calidad, fiabilidad y robustez del mismo; dentro del contexto o escenario previsto para ser utilizado. Las mismas permiten detectar la presencia de errores que generen salidas o comportamientos inapropiados durante su ejecución.(27)

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.(27)

Para lograr 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.(27)

Con el objetivo de comprobar el funcionamiento de las funcionalidades implementadas, fueron realizadas una serie de pruebas, a través de las cuales se pudo corroborar que el sistema responde positivamente a toda actividad que el usuario pueda realizar. Las pruebas aplicadas fueron pruebas de aceptación y pruebas de unicidad.

3.3.1 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.(27)

La validación del módulo se consigue mediante la realización de pruebas de caja negra y concentra su comportamiento en los RF del módulo.

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.(27)

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

Método de prueba de caja negra

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.(27)

A continuación se muestra una síntesis de los resultados obtenidos al aplicar las pruebas de caja negra con la utilización del método partición equivalente (ver Tabla 9), este método se aplicó a siete RF (RF 1, RF 9, RF 10, RF 11, RF 12, RF 13 y RF 15) con entrada de datos desde la interfaz de usuario y mostró resultados satisfactorios. Para una mayor profundización de los casos de prueba consultar el expediente de proyecto del Replicador de datos REKO relacionado con la solución propuesta. Los restantes casos de prueba se encuentran en los anexos del 44 al 49.

Tabla 9 Resultados de la prueba de caja negra sobre la HU 1

Caso de prueba	
Código: SC_1	**Historia de usuario:** 1
Nombre: Guardar la configuración para la resolución de conflictos de estructura.	
Descripción: prueba para la funcionalidad de guardar la configuración para la resolución de conflictos de estructura.	
Condiciones de Ejecución: se debe crear una configuración para la resolución de conflictos de réplica de estructura en la opción “Conflictos de estructura” del módulo “Configuración General”. Se debe seleccionar la opción “Automático” o “Manual”, así como las variantes para la resolución automática (“Ignorar cambio”, “Generar nombre de estructura”, “Aplicar cambio”, “Modificar estructura”, “Eliminar estructura”), para los diferentes tipos de conflictos tanto para un nodo ganador o un nodo no ganador.	
Respuesta del Sistema/Flujo Central: al seleccionar la opción “Manual” el sistema selecciona la opción y en el caso que se seleccione la opción “Automática” el sistema muestra la barra de título “Variantes para resolución de conflictos” para que el usuario determine las variantes para la resolución de conflictos para un nodo ganador o para un nodo no ganador.	
Resultado Esperado: el sistema guarda la configuración realizada por el usuario.	
Evaluación de la Prueba: Bien	

Fuente: elaboración propia del equipo de desarrollo

Resultados de las pruebas de caja negra

De manera general para las pruebas de caja negra se realizaron tres iteraciones. En una primera iteración al recibir un grupo de estructura replicable con 10 acciones, de ellas seis generaron conflictos, cuatro de creación de la estructura y dos de eliminación de la estructura. Se encontraron cuatro no conformidades (NC):

- En el módulo de Conflictos para la resolución manual no se mostraron las acciones referentes a la eliminación de la estructura.
- En la configuración general no se guardó la opción “Manual” en la “Resolución de modificación”.
- En la configuración general en la opción “Automática” en la “Resolución de eliminación” no se guarda la variante “Eliminar estructura”.
- No se monitoriza correctamente el nodo destino.

En una segunda iteración se resolvieron las cuatro NC detectadas en la primera iteración. En esta iteración se recibió un grupo de estructura replicable con 10 acciones, de ellas cinco generaron conflictos, tres de modificación de la estructura y dos de eliminación de la estructura. Se detectaron tres NC:

- En el módulo de Conflictos en la resolución de conflictos tipo “Eliminación de la estructura” para un nodo ganador con una configuración manual, muestra la interfaz cuando debería ignorar la acción automáticamente.
- En el módulo de Conflictos en la resolución de conflictos tipo “Modificación de la estructura” no muestra la acción que generó el conflicto.
- La restante NC responde a problemas de redacción.

En una tercera iteración se resolvieron las tres NC detectadas en la segunda iteración y se recibió un grupo de estructura replicable con 10 acciones, de ellas seis generaron conflictos, dos de creación de la estructura, tres de modificación de la estructura, y una de no existencia de la estructura, no se detectaron nuevas NC.

Con la aplicación de las pruebas de aceptación se validó que el módulo 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 módulo, donde se validó que el alcance es correcto.

3.3.2 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.(27)

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.(27)

Para la realización de la prueba de caja blanca el equipo de desarrollo optó por la técnica de camino básico debido a que proporciona el número mínimo de pruebas que se realiza por adelantado.

Técnica de caja blanca

La prueba del camino básico es una técnica de prueba de caja blanca que le permite al diseñador de casos de prueba obtener una medida de complejidad lógica de un diseño procedimental y que use esta medida como guía para definir un conjunto básico de rutas de ejecución. Los casos de prueba derivados para ejercitar el conjunto básico, deben garantizar que se ejecuta cada instrucción del programa por lo menos una vez durante la prueba.(27)

Los pasos a realizar para aplicar esta técnica son:

1. Representar el programa en un grafo de flujo: se utiliza para representar el flujo de control lógico de un programa.
2. Calcular la complejidad ciclomática: el valor calculado define el número de rutas independientes en el conjunto básico de un programa, y proporciona un límite superior para el número de pruebas que deben aplicarse, lo cual asegura que todas las instrucciones se hayan ejecutado por lo menos una vez.

3. Determinar el conjunto básico de rutas independientes: es cualquier ruta del programa que ingresa por lo menos un nuevo conjunto de instrucciones de procesamiento o una nueva condición.
4. Derivar los casos de prueba que fuerzan la ejecución de cada camino.(27)

A continuación se muestra el caso de prueba guiado por los pasos anteriores al método applyChange, el cual se localiza en la clase StructureNotExistsConflict. El objetivo de este método es ejecutar una sentencia en la BD que elimine las estructuras que tengan referencias, en el caso que el nodo local se comporte como un nodo no ganador. Se escogió este método por la importancia que tiene para la resolución de conflictos.

En la figura 8 se muestra el código fuente referente al método descrito anteriormente y en la figura 9 se aplica la técnica de camino básico, cuyo procedimiento se realizó manual y automático con la utilización de la herramienta Visustin[39] referente al código fuente presentado. En la tabla 10 se representa la complejidad ciclomática y posterior a ella se presenta el conjunto de caminos independientes.

[39] 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*

```
0  public ReplicableStructureAction applyChange(ReplicableStructureGroup replicableStructureGroup,
    ReplicableStructureAction replicableStructureActionConflict,MetadataManager metadataManager,
    MonitoringManager monitoringManager)
   {
1      ReplicableStructureAction replicableStructureAction = new ReplicableStructureAction();
2      try
       {
3          HibernateDialect dialect = DialectFactory.getDialect(metadataManager);
           Connection con = null;
           Statement stt = null;
           String dropSchema;
           String dropTable;
           String dropColum;
           String dropConstraint;
           con = metadataManager.getConnection();
           stt = con.createStatement();
4          if(GeneralConfig.isDropConflictAuto())
           {
5              if(replicableStructureActionConflict.getActionType().equals(ActionType.drop_schema))
               {
6                  dropSchema = DialectUtils.sentenceDropSchemaReference(dialect,
                           replicableStructureActionConflict.getSchemaName());
                   stt.executeUpdate(dropSchema);
                   replicableStructureAction = null;
               }
7              else if(replicableStructureActionConflict.getActionType().equals(ActionType.drop_table))
               {
8                  dropTable = DialectUtils.sentenceDropTableReference(dialect,
                           replicableStructureActionConflict.getTableName());
                   stt.executeUpdate(dropTable);
                   replicableStructureAction = null;
               }
9              else if(replicableStructureActionConflict.getActionType().equals(ActionType.drop_colum))
               {
10                 dropColum = DialectUtils.sentenceDropColumnReference(dialect, replicableStructureActionConflict.getTableName()
                           , replicableStructureActionConflict.getColumnName());
                   stt.executeUpdate(dropColum);
11             else if(replicableStructureActionConflict.getActionType().equals(ActionType.alter_add_primaryKey))
               {
12                 dropConstraint = DialectUtils.sentenceDropPkReference(dialect, replicableStructureActionConflict.getTableName()
                           , replicableStructureActionConflict.getOtherEspecification());
                   stt.executeUpdate(dropConstraint);
                   replicableStructureAction = null;
               }
           }
       }
13     catch (Exception e)
       {
14         e.printStackTrace();
       }
15     return replicableStructureAction;
16 }
```

Figura 8 Código fuente correspondiente al método applyChange de la clase StructureNotExistsConflict

Fuente: elaboración propia del equipo de desarrollo

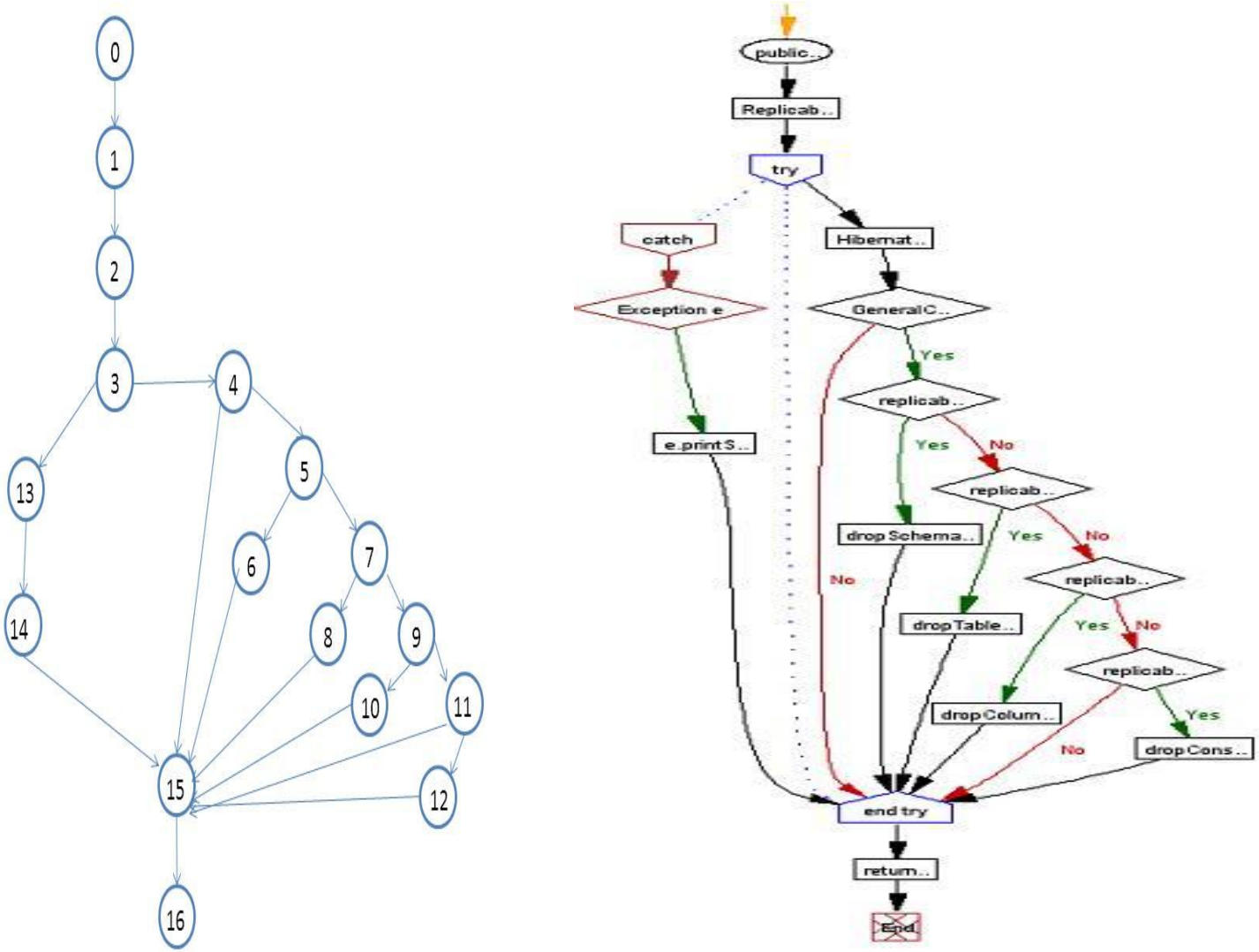


Figura 9 Flujo manual y automático correspondiente al método applyChange
Fuente: elaboración propia del equipo de desarrollo

Complejidad ciclomática:

Tabla 10 Complejidad ciclomática correspondiente al método applyChange

V(G) = R	V(G) = A - N + 2	V(G) = P + 1
R = 7 (regiones del grafo)	A = 22 (aristas) N = 17 (nodos)	P = 6 (nodos predicados)
	V(G) = 22 – 17 + 2	V(G) = 6 + 1
V(G) =7	V(G) = 7	V(G) = 7

Fuente: elaboración propia del equipo de desarrollo

Conjunto de rutas independientes:

Ruta 1: 0-1-2-3-13-14-15-16

Ruta 2: 0-1-2-3-4-15-16

Ruta 3: 0-1-2-3-4-5-6-15-16

Ruta 4: 0-1-2-3-4-5-7-8-15-16

Ruta 5: 0-1-2-3-4-5-7-9-10-15-16

Ruta 6: 0-1-2-3-4-5-7-9-11-12-15-16

Ruta 7: 0-1-2-3-4-5-7-9-11-15-16

A continuación se diseñan los casos de prueba que cubren los caminos independientes presentados.

Tabla 11 Casos de pruebas de los caminos independientes

No.	Entrada	Resultado
1.	Recibe como parámetro el objeto metadataManager vacío.	Lanza una excepción.
2.	Recibe todos los parámetros correctos pero la resolución es manual.	Retorna una acción vacía.
3.	Recibe todos los parámetros correctos pero la acción que está en conflicto es drop_schema.	Aplica correctamente la sentencia en la BD.
4.	Recibe todos los parámetros correctos pero la acción que está en conflicto es drop_table.	Aplica correctamente la sentencia en la BD.
5.	Recibe todos los parámetros correctos pero la acción que está en conflicto es drop_colum.	Aplica correctamente la sentencia en la BD.
6.	Recibe todos los parámetros correctos pero la acción que está en conflicto es	Aplica correctamente la sentencia en la BD.

	alter_add_primaryKey.	
7.	Recibe todos los parámetros correctos pero la acción que está en conflicto no es ninguna de las anteriores.	Retorna una acción vacía.

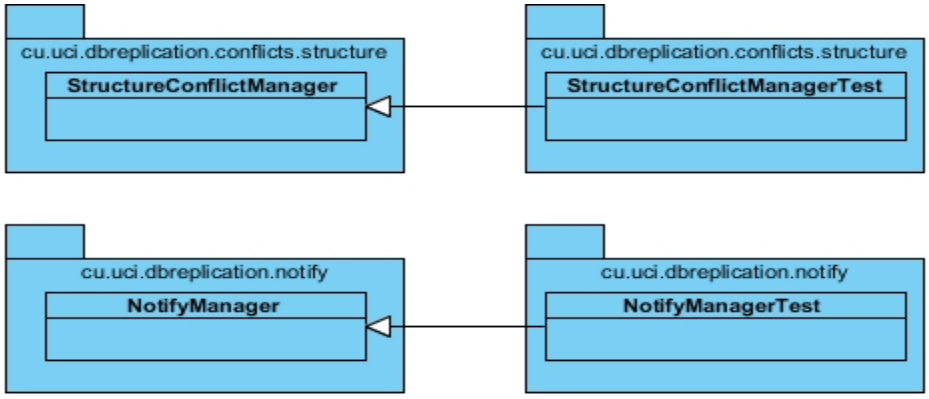
Fuente: elaboración propia del equipo de desarrollo

Framework JUnit

Para comprobar el resto de los métodos referentes a la solución se realizaron pruebas unitarias con la utilización de la técnica de caja blanca, las cuales posibilitan comprobar cada uno de los métodos por separado y definir si el software está listo y correctamente terminado. Mediante estas pruebas fue posible comparar la respuesta que debería obtenerse de un método con lo que realmente.(57)

Se escogió el *framework* JUnit para facilitar la implementación de pruebas de unidad, por su amplio uso y las facilidades que brinda para recolectar resultados de manera estructurada, reportes, relaciones entre las pruebas y la reutilización de código.(57)

Específicamente se empleó JUnit4, el que usa anotaciones de Java para identificar cuáles son los métodos, clases y *suites*[40] de prueba, facilitó la implementación de las pruebas realizadas. Las pruebas fueron diseñadas en relación con la herencia establecida entre las clases a probar, de tal forma que las funcionalidades encapsuladas en las clases de prueba fueron implementadas según los paquetes de las clases a probar como se muestra en la siguiente figura:



[40] Se refiere a un conjunto de piezas destinadas a ser tocadas en sucesión.

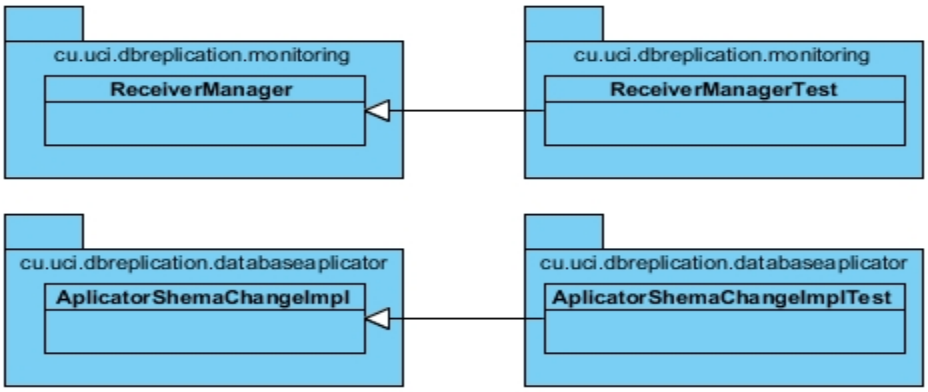


Figura 10 Clases de pruebas
Fuente: elaboración propia del equipo de desarrollo

Una vez definidas las pruebas, fueron aplicadas a través de una *suite* de pruebas definida en la clase TestRunnerAllTest, como se muestra en la siguiente figura:

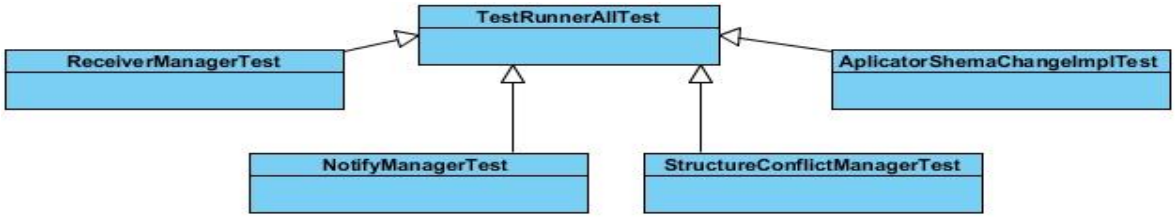


Figura 11 Suite de prueba
Fuente: elaboración propia del equipo de desarrollo

Los errores y fallos detectados en ejecuciones de la *suite* de pruebas fueron corregidos, y al final se obtuvo un resultado satisfactorio. Como se observa en la siguiente figura:

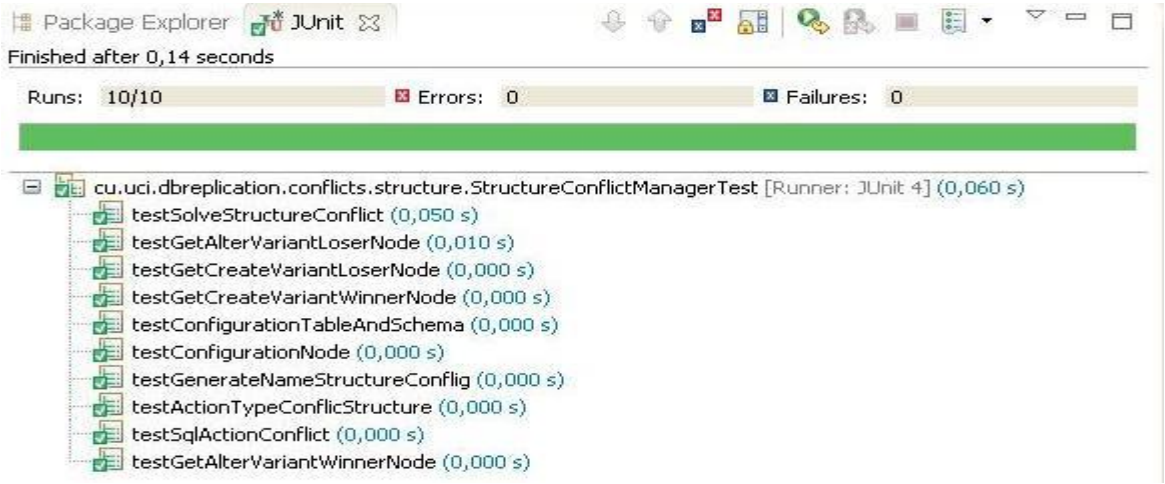


Figura 12 Vista de la ejecución de las pruebas del framework JUnit de la clase StructureConflictManagerTest
Fuente: elaboración propia del equipo de desarrollo

Los restantes resultados relacionados con la ejecución de las pruebas se pueden consultar en los anexos del 50 al 52.

Resultados de las pruebas de caja blanca

Las pruebas de caja blanca se aplicaron al código fuente del módulo, donde se comprobó el buen funcionamiento de los restantes RF (RF 2, RF 3, RF 4, RF5, RF 6, RF 7, RF 8 y RF 14), lográndose las respuestas deseadas, estas se pudieron corroborar con la ayuda del framework JUnit.

De manera general para las pruebas de caja blanca se realizaron tres iteraciones. En una primera iteración al recibir un grupo de estructura replicable con 10 acciones, cinco generaron conflictos, tres de creación de la estructura y dos de eliminación de la estructura, detectándose cuatro NC:

- No se persiste el grupo que se recibe de un nodo remoto en el caso que exista un grupo en conflicto.
- No se envió la notificación de actualizar la estructura del nodo remoto.
- No se realizó una correcta consulta a la DB.
- No se persiste la acción en conflicto.

En una segunda iteración se resolvieron las cuatro NC detectadas en la primera iteración. En esta iteración se recibió un grupo de estructura replicable con 10 acciones, tres generaron conflictos de modificación de la estructura. Se detectaron dos NC:

- En la resolución de conflictos de modificación de la estructura para un nodo ganador o para un nodo no ganador en las variantes “Generar nombre de estructura” y “Modificar estructura” no se tomó en cuenta las acciones que por su estructura no se puede cambiar el nombre.
- No se realiza un registro del proceso de resolución de conflictos de estructura.

En una tercera iteración se resolvieron las dos NC detectadas en la segunda iteración y se recibió un grupo de estructura replicable con 10 acciones, seis generaron conflictos, tres de creación de la estructura, dos de modificación de la estructura y una de no existencia de la estructura, no se detectaron nuevas NC.

3.4 Resultados obtenidos

Como resultados obtenidos después de realizar las pruebas al sistema, se evidencia que:

- El módulo mostró ser completamente funcional, se dio solución a todos los requisitos planteados.
- Todos los problemas de tratamiento de errores que fueron descubiertos durante el desarrollo de las pruebas arrojaron resultados satisfactorios, de esta manera se demuestra que el sistema cumple con cada uno de los requisitos establecidos.

3.5 Consideraciones del capítulo

Una vez realizada la implementación y las pruebas del sistema se obtuvieron las siguientes consideraciones:

- El módulo mostró ser completamente funcional, se dio solución a todos los requisitos funcionales planteados.
- Los componentes diseñados en el modelo de implementación permiten la organización del código fuente del módulo para la resolución de conflictos de estructura.
- A partir de la metodología AUP se diseñaron los casos de prueba que permitieron evaluar el funcionamiento del módulo desarrollado.
- La realización de las pruebas de software permitió detectar y corregir los errores durante la implementación de la solución al problema planteado.

CONCLUSIONES GENERALES

La investigación desarrollada y los resultados obtenidos permiten a los autores plantear las siguientes conclusiones finales:

- El uso de métodos teóricos y empíricos fue la guía en el inicio del desarrollo de la presente investigación, lo que facilitó el procedimiento para lograr la coherencia de los datos que se replican.
- El estudio de los sistemas existentes demostró que las herramientas homólogas no brindan solución al problema de la investigación, por lo que fue necesario desarrollar un módulo para la resolución de conflictos de estructura.
- El desarrollo del módulo aseguró la resolución de conflictos de estructura de manera automática y manual, por lo que se evitan incoherencias en los datos que se replican y mantiene la consistencia de las base de datos.
- Las pruebas realizadas validaron la coherencia de los datos en el proceso de réplica, lo que demostró la solidez de la implementación de la solución propuesta.

RECOMENDACIONES

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

- Incluir el mecanismo de resolución automática y manual de conflictos de estructura para los dialectos Oracle, Microsoft SQL Server y MySQL.

REFERENCIAS BIBLIOGRÁFICAS

1. COMPANIONI SARDIÑA, Y. A. R., ALBIN; HERNÁNDEZ SUAREZ,EYVIS; VELAZQUEZ VIZCAY,ADRIEL; NUÑEZ RIOS,MADIELENNIS; PÉREZ MATOS, LEISER. Reko: una solución de réplica para sistemas de bases de datos relacionales distribuidos. *Cuba: IV Simposio Informática y Comunidad.*, 2012, nº

2. SIERRA, M. *¿Qué es una Base de Datos y cuáles son sus principales tipos?* [Página web: Sitio oficial]. España: Aprender a programar, [Consultado el: noviembre de 2014]. Disponible en: http://www.aprenderaprogramar.com/index.php?option=com_attachments&task=download&id=500.

3. ROSA MARÍA MATO GARCÍA. *Diseño de bases de datos* Editorial Pueblo y 2009. 51 p. ISBN 978-959-13-1273-0.

4. DAVID BOLTON. *Definition of Database* [Página web: Sitio oficial]. About [Consultado el: noviembre de 2014]. Disponible en: http://cplus.about.com/od/glossar1/g/databasedefn.htm.

5. CARRANZA ATHÓ, F. *Bases de datos distribuidas. .* Perú: Perú: Escuela Académico Profesional de Informática, 2006.

6. COLECTIVO DE AUTORES, I. T. D. V. *Creación del esquema de la base de datos* [Página web: Sitio Oficial]. Mexico: Instituto Tecnológico de Veracruz, Última actualización: 02 de febrero de 2012. [Consultado el: noviembre de 2014]. Disponible en: http://www.prograweb.com.mx/tallerBD/0201Esquema.php.

7. DHARMA ROJAS. *Propuesta Metodológica para el Desarollo y la Elaboración de Estadísticas ambientales para los países de América Latina y el Caribe.* United Nations Publications, 2009. 36 p p. ISBN 9213227787, 9789213227787.

8. JOSE ANTONIO OCAMPO. *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. 18 p p. ISBN 9789213222836.

9. MAZILU, M. C. Database Replication. *Database Systems Journal*, 2010, vol. 1, nº 2, p. 33-38. Disponible en: http://www.dbjournal.ro/archive/2/2.pdf#page=34. ISSN 2069-3230.

10. URBANO, R. *Oracle Database Advanced Replication, 11g Release 2 (11.2)* [Página web: Sitio oficial]. Oracle®, Última actualización: junio 2013. [Consultado el: noviembre de 2014]. Disponible en: http://docs.oracle.com/cd/E11882_01/server.112/e10706.pdf.

11. MICROSOFT. *Replicación de SQL Server* [Página web: Sitio oficial]. Microsoft, Última actualización: octubre 2013. [Consultado el: noviembre de 2014]. Disponible en: http://msdn.microsoft.com/es-es/library/ms151198.aspx.

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

13. FERREIRA, J. E.; TAKAI, O. K., *et al.* Banco de Dados. *Instituo Federal do Espírito Santo, Vitória*, 2009, nº Disponible en: http://www.ime.usp.br/~mfinger/2011/mac426/BDSlides06branco.pdf.

14. TARANCÓN, J. P. Réplica de datos en Oracle. Integración con la arquitectura GRID. *Serie Científica*, 2008, vol. 1, nº 5, Disponible en: http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_2810_08.

15. CORY JANSSEN. *Modular Programming* [Página web: Sitio oficial]. Última actualización: 2010. [Consultado el: diciembre de 2014]. Disponible en: http://www.techopedia.com/definition/25972/modular-programming.

16. LEANDRO ALEGSA. *Diccionario de Informática y Tecnología. ¿Qué significa Módulo?* [Página web: Sitio oficial]. Argentina: Última actualización: 30 de junio de 2009. [Consultado el: diciembre de 2014]. Disponible en: http://www.alegsa.com.ar/Dic/modulo.php.

17. LONG, E. *SymmetricDS* [Página web: Sitio oficial]. SymmetricDS, JumpMind, Inc, Última actualización: 23-06-2014. [Consultado el: septiembre de 2014]. Disponible en: http://www.symmetricds.org/.

18. ENTERPRISEDB CORPORATION. *How to Setup Slony-I Replication with Postgres Plus* [Página web: Sitio oficial]. Estados Unidos: EnterpriseDB, Última actualización: 16-08-2011 [Consultado el: septiembre de 2014]. Disponible en: http://www.enterprisedb.com/resources-community/tutorials-quickstarts/all-platforms/how-setup-slony-i-replication-postgres-plus.

19. JAN WIECK. *Slony-I A replication system for PostgreSQL* [Página web.]. Estados Unidos: Afilias USA INC, Última actualización: 25-05-2013. [Consultado el: septiembre de 2014]. Disponible en: http://slony.info/images/Slony-I-implementation.pdf.

20. SOURCE FORGE. *Proposal to Open Source Daffodil Replicator* [Página web: Sitio Oficial]. Source Forge, Última actualización: 22 de mayo de 2014. [Consultado el: septiembre de 2014]. Disponible en: http://sourceforge.net/projects/daffodilreplica.

21. ORACLE. *Data Replication and Integration* [Página web:Sitio oficial]. Oracle, Última actualización: 2007. [Consultado el: septiembre de 2014]. Disponible en: http://www.oracle.com/technetwork/database/information-management/streams-fov-11g-134280.pdf.

22. SERVER, M. S. *Making Schema Changes on Publication Databases* [Página web.]. [Consultado el: septiembre de 2014]. Disponible en: http://msdn.microsoft.com/es-es.

23. SERVER, S. *SQL Server Replication* [Página web.]. Última actualización: 2014. [Consultado el: septiembre de 2014]. Disponible en: http://msdn.microsoft.com/en-us/library/ms151198.aspx.

24. ALFONSO VALDÉS, J. G. P., ÁNGELA GLORIA; PIMENTEL GONZÁLEZ, LUIS ALBERTO. Desarrollo del módulo para la transmisión de datos de gran tamaño para el sistema Reko. *Serie Científica*, 2011, vol. 4, nº 12, Disponible en: http://publicaciones.uci.cu/index.php/SC/article/view/490/486.

25. PIMENTEL GONZÁLEZ, L. A. H. S., EIVYS; BERMÚDEZ PEÑA,ROLANDO; PÉREZ ALFONSO,DAMIÁN; MENA RODRIGUEZ,LUIS EDGARDO; ALFONSO VALDÉS,JAVIER; GOMEZ PEÑA, ANGELA GLORIA Reko Replicador. *Cuba: Universidad de las Ciencias Informáticas.*, 2009, vol. 9, nº

26. MADIELENNIS NÚÑEZ DE LOS RÍOS, E. D. P. *Manual de usuario Replicador de datos REKO V4.0.* 2015, 97 p.

27. PRESSMAN, R. S. *Ingenieria del Software, Un Enfoque Práctico.* Editado por: Ed. España:: Editorial McGraw-Hil, 2005. 640 p p. ISBN 8448132149

28. ERVIN FLORES, J. L. C. *Metodologías Agiles Proceso Unificado Ágil (AUP)* [Página web: Sitio oficial]. La Paz, Bolivia: Universidad Unión Bolovariana carrera de Ingeniería de Sistemas, Última actualización: 08 de julio de 2009. [Consultado el: noviembre de 2014]. Disponible en: http://ingenieriadesoftware.mex.tl/63758_AUP.html.

29. SÁNCHEZ, T. R. *Metodología de desarrollo para la Actividad productiva de la UCI.* La Habana, Cuba: Universidad de las Ciencias Informáticas, 2014, [Consultado el: diciembre 2014]. 13 p.

30. BY KIM HAMILTON, R. M. *Learning UML 2.0.* O'Reilly, 2006. 286 p p. ISBN 0-596-00982-8.

31. VISUAL-PARADIGM. *Visual-Paradigm* [Página web: Sitio Oficial]. Lai Chi Kok Road, Kln, Hong Kong: Última actualización: 02-2011. [Consultado el: octubre de 2014]. Disponible en: http://www.visual-paradigm.com/.

32. OSCAR, S. *Visual Paradigm for Uml.* International Book Market Service Limited, 2013. 88 p p. ISBN 9786139166534.

33. WEITZENFELD, A. *Ingeniería de software orientada a objetos con UML, Java e Internet.* Thomson, 2005. 678 p p. ISBN 9789706861900.

34. PARI, J. *Manual de Instalación de SpringSourse Tool Suite My JBreak* [Página web]. Última actualización: 14 de febrero del 2012. [Consultado el: octubre de 2014]. Disponible en: http://www.slideshare.net/juliopari/spring-tool-suite-instalacion.

35. ORACLE. *Introduction to Java Platform, Enterprise Edition 7* [Página web:Sitio oficial]. Oracle, Última actualización: junio 2013. [Consultado el: octubre de 2014]. Disponible en: https://www.oracle.com/java/technologies/java-ee.html.

36. CHAHUICH, L. *Api De Java* [Página web: Sitio oficial]. SlideShare, Última actualización: 06 de febrero 2010. [Consultado el: octubre de 2014]. Disponible en: http://www.slideshare.net/lcahuich/1-1-3-api-de-java.

37. FUNDATION, T. A. S. *Apache ActiveMQ* [Página web: Sitio oficial]. THE APACHE SOFTWARE FUNDATION, Última actualización: 18 de julio 2014. [Consultado el: octubre de 2014]. Disponible en: http://activemq.apache.org/index.html.

38. THE APACHE SOFTWARE FUNDATION. *Apache Tomcat* [Página web: Sitio oficial]. THE APACHE SOFTWARE FUNDATION, Última actualización: 11 de noviembre de 2014. [Consultado el: noviembre de 2014]. Disponible en: http://tomcat.apache.org.

39. WALLS, C. Spring in Action. *Estados Unidos: Manning Publications Co*, 2008, vol. 3 ed., nº p. 426 p. [Consultado el: octubre 2014]. Disponible en: html://www.superjava-2009.googlecode.com/svn/trunk/TaiLieuSpring/Spring%20in%20action_slide.pdf. ISSN 9781935182351.

40. GUERRA, C. R. *Configuración JMS en ORACLE WEBLOGIC v11* [Página web: Sitio oficial.]. Lima: Oracle®, Última actualización: 10 de mayo de 2013. [Consultado el: octbre de 2014]. Disponible en: http://frameworksjava2008.blogspot.com/2012/05/configuracion-jms-en-oracle-weblogic.html.

41. PARSONS, D. *Foundational Java: Key Elements and Practical Programming.* Springer, 2012. 532 p, p. ISBN 1447124782.

42. FRANK ZAMMETTI. *Practical Dojo Proyects.* 2009 ed. Apress; , 2008. 500 p p. ISBN 1430210664.

43. GINESTA GILGERT, M. P. M., OSCAR. . *Bases de datos en PostgreSQL* [Página web: Sitio oficial]. Universidad Oberta de Catalunya, Última actualización: Febrero 2007. [Consultado el: octubre de 2014]. Disponible en: http://ocw.uoc.edu/computer-science-technology-and-multimedia/bases-de-datos/bases-de-datos/P06_M2109_02152.pdf.

44. LARMAN, C. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* 2005. 507 p p.

45. IAM SOMMERVILLE. *Requerimientos. En Ingeniería del Software.* Madrid: Pearson Education S.A, 2005. vol. 7 e.d, 712 p p. ISBN 84-7829-074-5.

46. KATERINE VILLAMIZAR SUAZA. *Definición de equivalencias entre historias de usuario y especificaciones en UN-LENCEP para el desarrollo ágil de software.* Facultad de Minas – Departamento de Ciencias de la Computación y de la Decisión. Universidad Nacional de Colombia, , 2013.

47. CARLOS REYNOSO, N. K. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. *UNIVERSIDAD DE BUENOS AIRES*, 2004, nº p. 73. Disponible en: http://www.willydev.net/descargas/prev/Estiloypatron.pdf.

48. GAMMA, E. Patrones de diseño: elementos de software orientado a objetos reutilizables, ed. P. *Educacion*, 2006, vol. 1, nº [Consultado el: 22 de febrero de 2015]. Disponible en: http://dspace.ucbscz.edu.bo/dspace/handle/123456789/565. ISSN 84-7829-059-1.

49. GUERRERO, C. A.; SUÁREZ, J. M.*, et al.* Patrones de Diseño GOF (The Gang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web. *Información tecnológica*, 2013, vol. 24, nº 3, p. 103-114. ISSN 0718-0764.

50. CECILIO ÁLVAREZ. *Patrones de Diseño (Active Record vs DAO)* [Página Web: Sitio Oficial]. Última actualización: 31 de julio de 2014. [Consultado el: 22 de febrero de 2015]. Disponible en: http://www.genbetadev.com/java-j2ee/patrones-de-diseno-active-record-vs-dao.

51. JACOBSON, I.; BOOCH, G.*, et al. El proceso unificado de desarrollo de software.* Addison Wesley Reading, 2000. vol. 7, ISBN 84-7829-036-2.

52. LARMAN, C. *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development, 3/e.* Pearson Education India, 2005. ISBN 8177589792.

53. ALCOCER PULUPA, D. E. y ORTIZ TAMAYO, 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.

54. JACOBSON, I.; BOOCH, G., *et al. El proceso unificado de desarrollo de software.* Pearson Educación, 2000. ISBN 9788478290369.

55. DEITEL, H. M. y DEITEL, P. J. *Cómo programar en C/C++ y Java.* Pearson Educación, 2004. ISBN 9702605318.

56. ABBAS, A. S. Re engineering of legacy software systems. 2015, nº

57. JOHNSON, R. *Expert One-on-One. J2EE Development without EJB.* Editado por: Published by Wiley Publishing, I., Indianapolis, Indiana. Canada: 2004. 577 p. ISBN 0-7645-5831-5.