Universidad de las Ciencias Informáticas

Facultad 6

Título: “Desarrollo del módulo para la transmisión de datos de gran tamaño para el sistema Reko.”

Trabajo de Diploma para optar por el título de
Ingeniero Informático

Autor(es): Angela Gloria Gomez Peña
Javier Alfonso Valdés

Tutor(es): Ing. Luis Alberto Pimentel González
Ing. Luis Enrique Ramírez Noy
Cotutor: Ing. Andrés Ballester Marsal

Ciudad Habana, Junio 2010.
“Año 52 de la Revolución”.

"El revolucionario verdadero está guiado por grandes sentimientos de amor"

Che

DECLARACIÓN 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 firmo la presente a los _____ días del mes de _____ del año _____.

_____ _____

Angela Gloria Gomez Peña Javier Alfonso Valdés

_____ _____

Ing. Luis Alberto Pimentel González Ing. Luis Enrique Ramírez Noy

_____

Ing. Andrés Ballester Marsal

DATOS DE CONTACTO

Ing. Luis Alberto Pimentel González
Ing. en Ciencias Informáticas, UCI 2007. Profesor Instructor.
E-mail: lapimentel@uci.cu

Ing. Luis Enrique Ramírez Noy
Ing. En Telecomunicaciones y Electrónica, CUJAE 2004. Profesor Instructor.
E-mail: noy@uci.cu

Ing. Andrés Ballester Marsal
Ing. en Informática, Universidad de Cienfuegos 2007. Profesor Instructor.
E-mail: aballester@uci.cu

AGRADECIMIENTOS

Al Comandante Fidel por crear el camino que he recorrido en estos 5 años para lograr mi sueño.

A Dios por darme las fuerzas para llegar hasta aquí.

A mis papis Anita y Sergito, quienes incondicionalmente me han amado y apoyado toda mi vida.

A mi abuelita Adis, por entregarme el espíritu guerrillero que la caracteriza.

A mi abuelito Sergio, que aunque ya no está, se sentiría muy orgulloso.

A mi tía Caridad, que como una madre me acogió estos 5 años.

A mi hermanita Tati por su cariño y locura que me acompañan siempre.

Al amor de mi vida (Nené), por amarme, comprenderme, ayudarme siempre y mostrarme que cada momento vale.

A mis mejores amigos: Clau, Albertini y Salvi, gracias por estar siempre a mi lado.

A Javier por ser un excelente compañero.

A mis tutores por su tiempo y comprensión.

En fin, a todos aquellos amigos y compañeros con los que he compartido todo este tiempo.

Angela

A la Revolución por darnos la oportunidad.

A mi familia, en especial a mis padres Maira Luisa y Juan, y a mi hermano Ernesto.

Al profesor Manuel Mariño.

A los tutores, por su apoyo y dedicación.

A todos aquellos que han sido parte, de una forma u otra, en este trayecto de 5 años.

Javier

A todos ustedes…Muchas Gracias.

DEDICATORIA

De Angela

A mi mami Anita, que tanta paciencia me ha tenido, que tanto amor me ha brindado y tanto tiempo me ha dedicado.

A mi papi Sergito, que siempre ha estado tan cerca, dándome las fuerzas para seguir adelante y el cariño para no extrañar tanto.

A mi abuelita Adis que tanto me mima cuando estoy en casa.

A mi tía Caridad, que me ha cuidado y adoptado como su hijita.

A mi novio y gran amor Nené, por ayudar con la presentación, con el documento; por el amor, la dedicación, la comprensión y la calma que me ha tenido. Gracias por todo Jose.

De Javier

A mis abuelos, Maria Luisa, Maria Antonia, Juan y Jose.

A mis padres.

RESUMEN

La Universidad de las Ciencias Informáticas cuenta con un replicador de datos, diseñado y construido en la propia Universidad, cuyo nombre es Reko. Este replicador ha sido utilizado en la práctica de proyectos reales tales como el Sistema de Gestión Penitenciaria de Venezuela; sin embargo, pese a su uso y aplicación, presenta la desventaja de no replicar con eficiencia datos de gran tamaño. Por lo que la Universidad decidió que era necesario brindar a Reko funcionalidades que le permitan dar solución a esta deficiencia.

El presente trabajo de diploma propone el diseño e implementación de un módulo que le brindará a Reko las facilidades de envío y recibo eficiente de datos de gran tamaño, de modo que cumpla con los requisitos funcionales y no funcionales establecidos por la Universidad.

PALABRAS CLAVE

base de datos, distribuidor, etiqueta, log, nodo, publicador, replicador, triggers.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Las Ciencias Informáticas están en continuo proceso evolutivo; por lo que su aplicabilidad en actividades que realiza el hombre es cada vez mayor. Los sistemas informáticos, parte significativa de esta ciencia, constituyen una de las principales aplicaciones y se fundamentan en la facilitación de determinadas acciones que comúnmente realizaría una persona o grupo.

En sus inicios, estos sistemas se encontraban solo de forma centralizada, pero la necesidad de extender sus prestaciones a distintas ubicaciones unido a la continua evolución de la Informática, dieron paso al surgimiento de los sistemas informáticos distribuidos, existiendo hoy en día una alta tendencia al desarrollo de los mismos. Algunos de estos sistemas plantean en su arquitectura el uso de varios servidores de bases de datos separados geográficamente, por lo que es esencial mantener una actualización y sincronización adecuadas para lograr la coherencia y la consistencia de la información.

Las soluciones comerciales, referentes a la temática de replicadores de bases de datos, son muy costosas y las que existen en software libre están incompletas y son muy complejas de manejar. La Universidad de las Ciencias Informáticas, desarrolló un software replicador nombrado Reko; el cual tiene como objetivo brindar una herramienta multiplataforma que le permita al usuario cubrir las necesidades fundamentales de replicación tales como sincronización, transferencia de datos entre diversas localizaciones y la centralización de la información en una única localización, la protección y la recuperación, así como cubrir otras necesidades relacionadas con la distribución de datos entre los gestores más populares.

El manejo de la distribución de la información es la principal función de un replicador, por lo que debe de ser capaz de manipular todo tipo de información; sin embargo, Reko carece de mecanismos efectivos para efectuar operaciones de réplica con datos de gran tamaño.

Surgiendo como **Problema Científico**: ¿Cómo realizar réplicas de datos de gran tamaño, empleando el software Reko?

El problema planteado se enmarca en el **Objeto de Estudio**: La replicación de datos de gran tamaño entre bases de datos relacionales.

Delimitado por el **Campo de Acción**: La replicación de datos de gran tamaño en el software Reko.

Para dar solución al problema planteado se define como **Objetivo General**: Desarrollar un módulo para Reko, que permita realizar réplicas de datos grandes.

Desglosándose en los **Objetivos Específicos**:

1. Diseñar un módulo que le permita a Reko realizar réplicas de datos grandes.
2. Implementar el módulo diseñado.
3. Integrar el módulo al software Reko.
4. Realizar pruebas exploratorias al módulo.

Para alcanzar el objetivo propuesto, se realizarán las siguientes **Tareas de la Investigación**:

1. Análisis del estado del arte acerca de aplicaciones y protocolos en el envío de grandes ficheros de datos en diferentes replicadores desarrollados.
2. Elaboración del modelo conceptual.
3. Definición de los requisitos funcionales y no funcionales de la aplicación.
4. Desarrollo del diagrama de clases del diseño.
5. Implementación del diagrama de clases del diseño.
6. Diseño y ejecución de las pruebas exploratorias.

El trabajo consta de introducción, cuatro capítulos, conclusiones, recomendaciones, bibliografía y anexos. En el **Capítulo 1 Fundamentos Teóricos de la Investigación:** Se realiza un pequeño bosquejo sobre la réplica de datos, lo cual incluye el estudio de algunos replicadores existentes, así como la descripción de las principales características de Reko. Se selecciona la metodología de desarrollo a utilizar y las herramientas empleadas en el desarrollo de la solución.

En el **Capítulo 2 Características del Sistema:** Se profundiza en el funcionamiento del software Reko. Se presenta el diagrama del modelo conceptual propuesto y se definen los requerimientos a implementar en el módulo.

En el **Capítulo 3 Diseño del Sistema:** Se realiza una descripción de los estilos arquitectónicos y patrones de diseño, desarrollo de los diagramas de clases del diseño así como la realización de los diagramas de secuencia, el modelo de despliegue y las descripciones de las principales clases.

En el **Capítulo 4 Implementación y Pruebas**: Se muestran los diagramas de componentes que se definieron durante la implementación de la aplicación, se presentan implementaciones relevantes y se realizan validaciones teóricas y funcionales a la solución propuesta.

CAPÍTULO 1: FUNDAMENTOS TEÓRICOS DE LA INVESTIGACIÓN

1.1 Introducción

En el presente capítulo, se abordan temas referentes a la necesidad de replicar información almacenada en bases de datos ubicadas en distintas locaciones; se abordan conceptos fundamentales relacionados con la replicación. Se realiza una breve descripción de la metodología y herramientas de desarrollo utilizadas, así como los roles y artefactos generados durante el desarrollo.

1.2 Réplica

La replicación es el proceso de copiar y mantener los objetos de base de datos, dígase tablas, en múltiples bases de datos que componen un sistema de bases de datos distribuidas. Los cambios aplicados en una locación, son capturados y almacenados localmente antes de ser transmitidos y aplicados en cada una de las ubicaciones remotas. [1]

La réplica de datos tiene diferentes clasificaciones atendiendo a los siguientes criterios:

- Ambiente de replicación:
  - **Maestro-esclavo** (master-slave)**:** En este caso la replicación se realiza en un único sentido: desde un nodo maestro a uno o varios esclavos.
  - **Multi-Maestro** (multi-master)**:** Se configuran varios nodos como maestros, que pueden replicar entre sí. [1]
- Forma de transmitir los cambios en el entorno de replicación:
  - **Síncrona:** Los cambios ejecutados en un nodo maestro son aplicados instantáneamente en el nodo origen y los nodos destino dentro de una misma transacción. En caso de no poder ejecutarse la acción en cualquiera de los nodos, la transacción completa es retrotraída en todos los nodos. Requiere una alta disponibilidad de recursos de red.
  - **Asíncrona:** Los cambios ejecutados en una tabla son almacenados y enviados posteriormente al resto de los nodos del entorno cada ciertos intervalos de tiempo. [1]
- Forma de capturar y almacenar los cambios a replicar:
  - **Basada en triggers:** Se crean una serie de triggers en la base de datos, que permiten capturar las operaciones de inserción, actualización y eliminación (DML) realizadas sobre las tablas a replicar.
  - **Basada en logs:** Se sostiene en la lectura de logs de cambios. [1]

1.3 Replicadores Comerciales

El desarrollo de estas poderosas herramientas, ha ido en aumento a medida que ha pasado el tiempo, y actualmente en el mercado existe una amplia gama de esta clase de software.

Partiendo de los requerimientos básicos para dar solución a la problemática planteada, dígase necesidad de un replicador multi-maestro, de replicación asíncrona y basado en triggers, capaz de replicar ficheros binarios de gran tamaño, se realizó un estudio. Siendo el primer criterio de selección empleado el hecho que pueda ser utilizado de forma gratuita.

A continuación, algunas de las herramientas estudiadas:

Daffodil Replicator v2.1 [2]**:** Es una herramienta implementada sobre Java, de código abierto para la integración, migración y protección de datos en tiempo real. Permite bi-direccionar datos de replicación y sincronización entre bases de datos homogéneas y heterogéneas, incluyendo Oracle, MySQL, Daffodil DB, PostgreSQL, entre otras. Incluye además:

- La capacidad de detectar y permitir resolución de conflictos.
- Independencia de la plataforma.
- Soporte para la replicación de datos de gran tamaño (Clob, Blob).

La principal limitante detectada en el estudio de este replicador es que no es capaz de replicar en un escenario multi-maestro.

Tungsten Replicator v2.0 [3]**:** Es un sistema Open Source de replicación maestro-esclavo. Su modo de replicación es basado en logs; radicando sus principales características en:

- Soporte para replicación entre bases de datos heterogéneas.
- Bases de datos soportadas son MySQL, Oracle y PostgreSQL.
- Es independiente de la plataforma.
- Replica entre bases de datos con estructuras iguales.
- Replica archivos de gran tamaño.

Pese a sus características, este replicador no es capaz de replicar en un escenario multi-maestro y la forma de captura y almacenamiento de los cambios es mediante logs, constituyendo este punto otra limitante.

DBReplicator [4]**:** Es un software de replicación multi-maestro, abierto para la red de aplicaciones Java. Las principales características de DBReplicator son las siguientes:

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

A pesar de las características mencionadas, tampoco responde a las necesidades planteadas pues no es capaz de replicar datos de gran tamaño (Clob, Blob, Bytea, etc.).

SymmetricsDS [5]: Es una solución de sincronización y replicación de datos empleando modo asíncrono.

- Brinda una interfaz web para la configuración de la réplica.
- Soporta los gestores MySQL, Oracle, SQL Server y PostgreSQL.
- Basa su replicación en disparadores que capturan los cambios en la base de datos.
- Puede instalarse como una aplicación web independiente o ser embebida dentro de una aplicación desarrollada en Java.

Aún utilizando el modo asíncrono como forma de transmitir los cambios y el uso de triggers para la captura y almacenamiento de los mismos, no cumple con los requerimientos planteados pues no es capaz de replicar datos de gran tamaño (Clob, Blob, Bytea, etc.).

1.3.1 Resultados

El estudio realizado arrojó resultados poco satisfactorios, pues los replicadores estudiados no cumplían con los requerimientos básicos planteados; sin embargo, por separado eran capaces de satisfacer algunos de estos requerimientos. Esta situación condujo a la alternativa de brindar a Reko las funcionalidades necesarias para ser empleado en el contexto del problema planteado, ya que este es un software que cumple con la mayoría de los requerimientos necesarios, además de poseer la ventaja de ser un producto desarrollado por la Universidad de las Ciencias Informáticas, de código abierto, fácil de configurar y de buenas prestaciones.

1.4 Características del Replicador Reko

Actualmente, Reko cuenta con un conjunto de componentes que permiten la replicación de datos, entre bases de datos distribuidas. Cada uno de estos componentes, que a continuación se exponen, cumple con una función específica, permitiendo al software cumplir su cometido. (Anexo #1)

- *Núcleo:* Maneja las configuraciones fundamentales del software y agrupa las principales funcionalidades de procesamiento de información.
- *Capturador de Cambios:* Captura los cambios que se realizan sobre la Base de Datos y se los entrega al Distribuidor de Datos.
- *Aplicador de Cambios:* Ejecuta sobre la Base de Datos los cambios que sean replicados hacia esta.
- *Distribuidor de Datos:* Determina para dónde debe ser enviado cada cambio realizado en la Base de Datos, los envía y es responsable que cada cambio llegue a su destino.
- *Servidor JMS:* Servidor de Mensajería bajo la especificación Java Message Service, es utilizado como punto intermedio en la distribución de la información enviada bajo JMS.
- *BD Local de Réplica:* Es utilizada para guardar las configuraciones propias de la réplica, las acciones sobre la Base de Datos que han dado conflicto al aplicarse y las acciones o transacciones que no han podido llegar a su destino.
- *Consola Web de Administración:* Permite realizar las configuraciones principales del software como el registro de nodos, configuración de las tablas a replicar, datos a replicar y el monitoreo del funcionamiento del software.
- *Base de datos:* Es la Base de Datos que se está replicando, el software de réplica envía los cambios que se realizan sobre ella y aplica los cambios que provienen de otros nodos de réplica. [6]

Todos estos componentes en acción, permiten a este software llevar a cabo el proceso de replicación; en los cuales, varía el número de Servicios de Réplica, así como el número de Bases de Datos que intervendrán en la replicación. (Anexo #2)

Reko, posee un conjunto de características y/o funciones tales como:

- Soporta la replicación de transacciones a través de Hibernate y la replicación de acciones a través de triggers; se integra con los gestores de bases de datos Oracle y PostgreSQL.
- Provee la facilidad de seleccionar el subconjunto de tablas y datos a ser replicados mediante la definición de filtros aplicables a las tablas y la selección de usuarios propios del gestor.
- El mecanismo de registro entre nodos de replicación se basa únicamente en un identificador (id) del nodo lo que permite la abstracción de los datos físicos de cada nodo como son el IP y el protocolo de comunicación.
- Los nodos de replicación manejan credenciales entre ellos para verificar la autenticidad de los datos transferidos.
- Permite conocer el estado de los datos transmitidos, puede realizarse en tiempo real a través de la Web así como dar un seguimiento al funcionamiento interno del mecanismo.
- La administración y configuración de la réplica se realiza a través de una interfaz Web, por lo que es administrable vía remota usando un navegador.
- El software de réplica puede ser instalado en cualquier sistema operativo y no necesita de un ambiente gráfico en el servidor para su funcionamiento.
- Detecta errores de conexión. Mantiene los datos de réplica en un estado estable en caso de desconexión. Al restablecerse la conexión, automáticamente sincroniza los datos entre las bases de datos.
- La transferencia de datos de replicación puede realizarse a soporte para replicación sobre TCP/IP, HTTP o por ficheros de forma manual. [6]

Reko. Nuevas Funcionalidades

El software de réplica de datos, Reko, fue diseñado y construido por un grupo de desarrolladores pertenecientes a la Universidad de las Ciencias Informáticas; y liberado el 2 de noviembre de 2009, en su versión 1.0, luego de ser aprobado por los Laboratorios de Pruebas y el grupo de Calisoft (Centro para la Excelencia en el Desarrollo de Proyectos Tecnológicos).

Reko cumple perfectamente con los requerimientos de replicación, así como estándares de calidad. Sin embargo, no es capaz de brindar mecanismos efectivos para la replicación de datos de gran tamaño; por lo que se decidió mejorar esta solución de software, desarrollando un nuevo módulo que le permita replicar efectivamente este tipo de datos, lo cual se puede lograr utilizando los servicios de un servidor FTP.

1.5 Metodologías y Herramientas

Para lograr el alcance de los objetivos trazados es necesario definir la metodología y herramientas a utilizar durante todo el proceso de desarrollo. Por lo que a continuación se describirán la metodología a seguir y las herramientas a utilizar.

1.5.1 Metodología de Desarrollo de Software Seleccionada

En la actualidad existe un gran número de metodologías y herramientas capaces de brindar todo tipo de soluciones; esta situación es lo que conduce a la problemática a la cual se enfrentan los desarrolladores de hoy. La selección de la metodología a emplear, propone los roles que el equipo de desarrollo deberá cubrir, las actividades que debe cumplir cada uno de ellos, los artefactos que serán generados de cada tarea, así como el registro de cada detalle de la información que se irá generando.

Las metodologías y técnicas existentes para el desarrollo de productos de software, son analizadas y definidas en la arquitectura de cada proyecto. Para ello se realiza un estudio de algunas metodologías existentes y se realiza la selección de la que será empleada. A continuación se caracterizan dos metodologías estudiadas.

RUP: es una metodología robusta que genera un gran número de artefactos y requiere de varios roles para el desarrollo de un proyecto, también es recomendado su empleo cuando se trata de proyectos de gran tamaño.

XP, sin embargo, es una metodología ágil, recomendada para ser usada en el desarrollo de proyectos de pequeñas dimensiones. Genera pocos artefactos y no requiere el desempeño de un gran número de roles.

Debido a lo antes expuesto y a las características del módulo que se desea desarrollar, donde interviene un pequeño grupo de trabajo y no son necesarios varios de los roles y artefactos propuestos por las metodologías antes descritas; siendo el módulo abarcador, sin llegar a constituir un proyecto de grandes dimensiones, se decidió hacer uso de la metodología previamente definida para la elaboración y construcción del software Reko: Open Up.

Open Up

Open Up/Basic en un framework de procesos de desarrollo de software de código abierto, construido sobre una donación realizada por IBM del Basic Unified Process y liberado por el Eclipse Process Framework (EPF). Permite abordar ágilmente el desarrollo de proyectos pequeños y tiene un enfoque centrado en el cliente con iteraciones cortas. [7]

Este proceso de desarrollo unificado está basado en las mejores prácticas de RUP, permitiendo de esta manera que se puedan desarrollar artefactos ligeros apropiados, utilizando el Unified Modeling Language (UML) e incrementando de esta forma las probabilidades de éxito en función de calidad, costo, tiempo y alcance.

Debido a que está basada en RUP, aplica un enfoque iterativo e incremental dentro de su estructura del ciclo de vida y puede adaptarse para desarrollar diversos tipos de proyectos. Por demás, es un proceso iterativo del desarrollo del software que es mínimo, completo, y extensible. [8]

El proceso es mínimo pues solamente el contenido fundamental es incluido; es completo porque puede ser manifestado todo el proceso para construir un sistema y es extensible pues puede ser utilizado como fundamento sobre el cual el contenido del proceso se puede agregar o adaptar según lo necesitado.

Open Up. Roles y Artefactos

Siguiendo el ciclo de vida que propone Open Up, para un proyecto de desarrollo de software, que incluye las fases de Inicio, Elaboración, Construcción y Transición; se desempeñarán los siguientes roles, que obtendrán sus respectivos artefactos:

Analista: Durante la fase de Inicio, en el Modelamiento del Negocio, su misión será definir la visión del sistema; describirá el problema que se presenta y las características del sistema según los requerimientos de los interesados. Estas actividades propiciarán que obtenga los artefactos correspondientes al Glosario de Términos y el Documento Visión.

En el proceso de Administración de Requerimientos, será el encargado de identificarlos y luego refinarlos; produciendo como salida los Casos de Uso del Sistema.

Probador: Durante la Fase de Inicio, en la Administración de Requerimientos, será el encargado de crear y desarrollar los Casos de Prueba, así como identificar los Datos de Prueba con los que se validarán los requerimientos que serán probados.

Durante la Fase de Elaboración, partiendo de los Casos de Prueba, implementará uno o varios artefactos de prueba para permitir la validación del sistema y luego ejecutará las pruebas para determinar la calidad del producto.

Desarrollador: Durante la Fase de Elaboración será el encargado de diseñar la solución. Partiendo de la Arquitectura, los Requerimientos de Soporte y los Casos de Uso, obtendrá un Diseño que dará paso para luego implementar las Pruebas de Desarrollador, siendo este el artefacto resultante de esta actividad.

Luego implementará la solución obteniendo como salidas el Build y la Implementación. Posteriormente ejecutará las Pruebas de Desarrollador para verificar los resultados y obtendrá los Resultados de Pruebas de Desarrollador.

1.5.2 Lenguaje de Modelado

Lenguaje Unificado de Modelado (UML)

"El Unified Modeling Language (UML) es un lenguaje estándar que permite especificar, visualizar, construir y documentar todos los elementos que forman un sistema de software orientado a objetos". Por lo que su propósito consiste en elaborar los artefactos de un sistema a través de las distintas etapas de su ciclo de vida, principalmente durante el análisis y el diseño del mismo. [9]

Modelar con UML trae consigo una serie de beneficios a los programadores, pues mediante él es posible establecer un conjunto de requerimientos y estructuras necesarias para plasmar un sistema de software previo a la escritura del código. "*Aunque UML es un lenguaje, éste posee más características visuales que programáticas"*, facilitando así el entendimiento común entre los miembros de un equipo de trabajo, dígase, analistas, diseñadores, programadores, etc. [10]

El tiempo invertido en el desarrollo de la arquitectura se minimiza, la trazabilidad y documentación del proyecto se realizan de forma ordenada y guiada por los casos de uso, pero lo más importante es la notable efectividad y productividad que se consigue en labores de diseño arquitectónico y mantenimiento, haciendo uso de UML frente a la realización de las mismas tareas, en ausencia de modelos. [11]

1.5.3 Herramientas CASE

Visual Paradigm

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño Orientado a Objetos (OO), construcción, pruebas y despliegue. Permite la elaboración de todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación de diferentes extensiones (.Pdf, .Doc, etc.).

Facilita el modelado colaborativo con Sistema de Versiones Concurrentes (Concurrent Versions System - CVS) y Subvertion (control de versiones), y la realización del proceso de ingeniería así como ingeniería inversa (proceso ingenieril en el que se obtienen modelos conceptuales a partir de los artefactos de software como código fuente, ejecutables, binarios y ficheros intermedios). Brinda soporte para el mapeo de objeto relacional (Object-Relational Mapping – ORM) y facilidades para la generación de bases de datos. Es una herramienta colaborativa, pues soporta múltiples usuarios trabajando sobre el mismo proyecto. [12]

Presenta licencia gratuita y comercial. Es fácil de instalar y actualizar, y compatible entre ediciones. Debido a todo lo antes mencionado, se puede concluir que Visual Paradigm se caracteriza por su robustez, usabilidad y portabilidad, por lo que se decidió adoptar como herramienta CASE para realizar el modelado de todo el módulo.

1.5.4 Lenguaje de Programación

Lenguaje Java

Java es un lenguaje de programación OO desarrollado por Sun Microsystems, a principio de los años 90´s, e independiente de la plataforma. Es un lenguaje de propósito general, lo cual permite crear diversos tipos de aplicación con él; radicando su mayor éxito en Internet, con el uso de los Applets, las aplicaciones cliente – servidor, o las Java Server Pages. [13]

Se distingue por ser capaz de gestionar la memoria automáticamente; no permite el uso de técnicas de programación inadecuadas; posee mecanismos de seguridad incorporados, los cuales limitan el acceso a recursos de las máquinas donde se ejecuta; incorpora herramientas de documentación; es multi-hilos (multithreading). Debido a estas características es calificado como un lenguaje de programación robusto.

1.5.5 Herramientas del Entorno de Desarrollo

Eclipse IDE

Desarrollado inicialmente por IBM como el sucesor de su familia de herramientas para VisualAge. Siendo ahora desarrollado por la Fundación Eclipse, una organización independiente, sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse es una plataforma universal o entorno de desarrollo integrado de código abierto, con una arquitectura abierta y basada en plug-ins; lo cual permite integrar diversos lenguajes sobre un mismo Entorno Integrado de Desarrollo (Integrated Development Environment – IDE) e introducir otras aplicaciones accesorias. [14]

Es una herramienta multiplataforma, soportada por sistemas operativos tales como:

- Linux
- Windows
- Solaris 8 (SPARC/GTK 2)

Dispone de un Editor de Texto con resaltado de sintaxis. La compilación es en tiempo real. Contiene pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes (wizards) para creación de proyectos, clases, pruebas, etc., y refactorización.

ActiveMQ v2.0

Es un popular, poderoso y rápido cliente de mensajería de código abierto. Se encuentra distribuido bajo la licencia Apache en su versión 2.0; soporta gran variedad de lenguajes de programación (Java, C, C++, C#, Paython, etc.). Puede ser utilizado como un proveedor de JMS en la memoria, ideal para pruebas de unidades de JMS.

ProFTPd

ProFTPd es un software servidor FTP seguro y estable, siempre y cuando sea configurado correctamente, con licencia GPL y cuenta además, con una robusta documentación. Tiene la capacidad de ser enjaulado en dependencia del sistema de archivos que haya por debajo.

Puede ser ejecutado como un demonio propio o como un servicio más de Inetd. Es capaz de trabajar sobre IPv6; posee un diseño modular, lo cual le permite escribir extensiones como cifrado SSL/TLS, RADIUS, LDAP o SQL como módulo. Funciona sobre sistemas operativos tales como: Linux for IBM S/390, Linux, Solaris, Gentoo entre otros.

Este servidor FTP será utilizado para probar las implementaciones realizadas en el módulo para la trasmisión de datos de gran tamaño.

1.6 Conclusiones

- Para el desarrollo del módulo se definió que será utilizada la Metodología Open Up, pues se puede adaptar fácilmente a proyectos pequeños. Es aplicable a un amplio conjunto de plataformas y aplicaciones de desarrollo y además es ágil, iterativa e incremental.

- El lenguaje de modelado a utilizar será UML, ya que permite especificar, visualizar, construir y documentar todos los elementos o artefactos durante todo el proceso de desarrollo del software.

- Se utilizará Visual Paradigm, versión 6.4, como herramienta CASE para realizar el modelado, así como materializar otras funcionalidades que este ofrece.

- El lenguaje de programación será Java, pues es un lenguaje de programación Orientado a Objetos, que permite implementar multi-hilos, función necesaria para el presente software.

- El IDE de desarrollo será Eclipse en su versión 3.4.0. Es multiplataforma, lo cual es convenientemente cómodo. Además de poseer otras características que lo convierten en una herramienta eficiente.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

2.1 Introducción

En el presente capítulo, se realiza una descripción de la solución a desarrollar, su integración con el software existente, los requisitos funcionales y no funcionales tenidos en cuenta, y casos de uso que se implementarán.

2.2 Descripción Detallada del Proceso a Automatizar

Mantener actualizadas las bases de datos de un sistema informático distribuido, es la tarea fundamental de un replicador de datos.

Actualmente Reko, es capaz de replicar utilizando un servidor JMS, el ActiveMQ, que actúa como punto intermedio en la distribución de la información. Al ser JMS un estándar de mensajería, se dificultan las operaciones de envío y recibo de datos de gran tamaño, pues este funciona con una cola de solicitudes de envío, y hasta que una determinada solicitud no ha sido completada (recibida), no se inicia el envío de la próxima.

El envío de una solicitud puede ser obstaculizado por inestabilidad en las conexiones, por un fallo eléctrico, etc. En caso de ocurrir alguna de estas afectaciones, la transacción es interrumpida, manteniendo los datos estables, y se iniciará nuevamente de manera automática cuando las comunicaciones estén restablecidas. Por lo tanto una transacción puede estar en cola por tiempo indefinido.

Ante esta limitante, se requiere brindar a Reko una solución para que sea capaz de replicar de forma eficiente datos de gran tamaño.

2.3 Funcionamiento del Software de Replicación Reko

El software Reko, se ejecuta en un entorno de replicación **Multi-Maestro** teniendo una instancia en cada nodo, los cuales pueden estar agrupados por Etiquetas. Desde cada nodo se permite configurar la réplica hacia otro nodo o hacia una Etiqueta, además de poder realizarse distintas configuraciones según los nodos y etiquetas que se definan.

Las configuraciones independientes de las tablas que se desean replicar, son registradas por configuración. Cada una de estas tablas puede ser configurada para replicar en dependencia de la acción que se efectúe sobre ella (inserción, actualización, eliminación). Además pueden definirse filtros SQL para determinar por cada acción si se replicará o no el cambio.

Cada determinados intervalos de tiempo, el **Capturador de Cambios**, registra los cambios realizados sobre la base de datos y crea un objeto de tipo ReplicableGroup, que contendrá un grupo de objetos de la clase ReplicableAction, cada uno de estos objetos almacena toda la información necesaria para replicar los cambios. A partir de estos objetos, las configuraciones registradas y los filtros asociados a estas, se determinan los destinos hacia donde se enviará el ReplicableGroup. Para la distribución de los ReplicableGroup, el **Distribuidor de Datos**, los envía como mensajes JMS utilizando un servidor de mensajería (JMS ActiveMQ), encargándose este, de la entrega de los ReplicableGroup y del acuse de recibo de los mismos.

Una vez recibido un ReplicableGroup en cada nodo destino, pasa al **Aplicador de Cambios** para ejecutar las acciones que contiene en la base de datos destino.

Los ReplicableGroup, poseen mecanismos de chequeo de envío, confirmación de recepción y registro en una base de datos local; estos garantizan la integridad y persistencia de los datos que se replican, ante posibles eventualidades.

2.4 Modelado del Dominio

2.4.1 Descripción de las Clases del Modelo de Dominio

- **Consola _Web_Administración:** Concepto que representa la interfaz web de la aplicación, mediante la cual se podrán realizar las principales configuraciones del software.
- **Núcleo:** Concepto que manejará las principales configuraciones del software.
- **Distribuidor_Datos:** Concepto que determinará el destino de cada configuración de cambio.
- **Capturador_Cambios:** Concepto encargado de capturar los cambios realizados a la base de datos y de entregarlos al Distribuidor.
- **Servidor_JMS:** Concepto que representa un servidor de mensajería, utilizado como punto intermedio para la distribución de la información enviada.

- **Aplicador_Cambios:** Concepto que ejecutará sobre la Base_Datos los cambios que sean replicados hacia ella.
- **Base_Datos_Local_Réplica:** Concepto que guardará las configuraciones propias de la réplica, así como acciones sobre la Base_Datos que han provocado conflicto al aplicarse y transacciones que no han llegado a su destino.
- **Base_Datos:** Concepto que representa la Base de Datos que se está replicando. Los cambios ejecutados sobre ella serán enviados así como serán aplicados otros cambios provenientes de otros nodos de réplica.

2.4.2 Diagrama de Clases del Modelo de Dominio

Modelo de Dominio

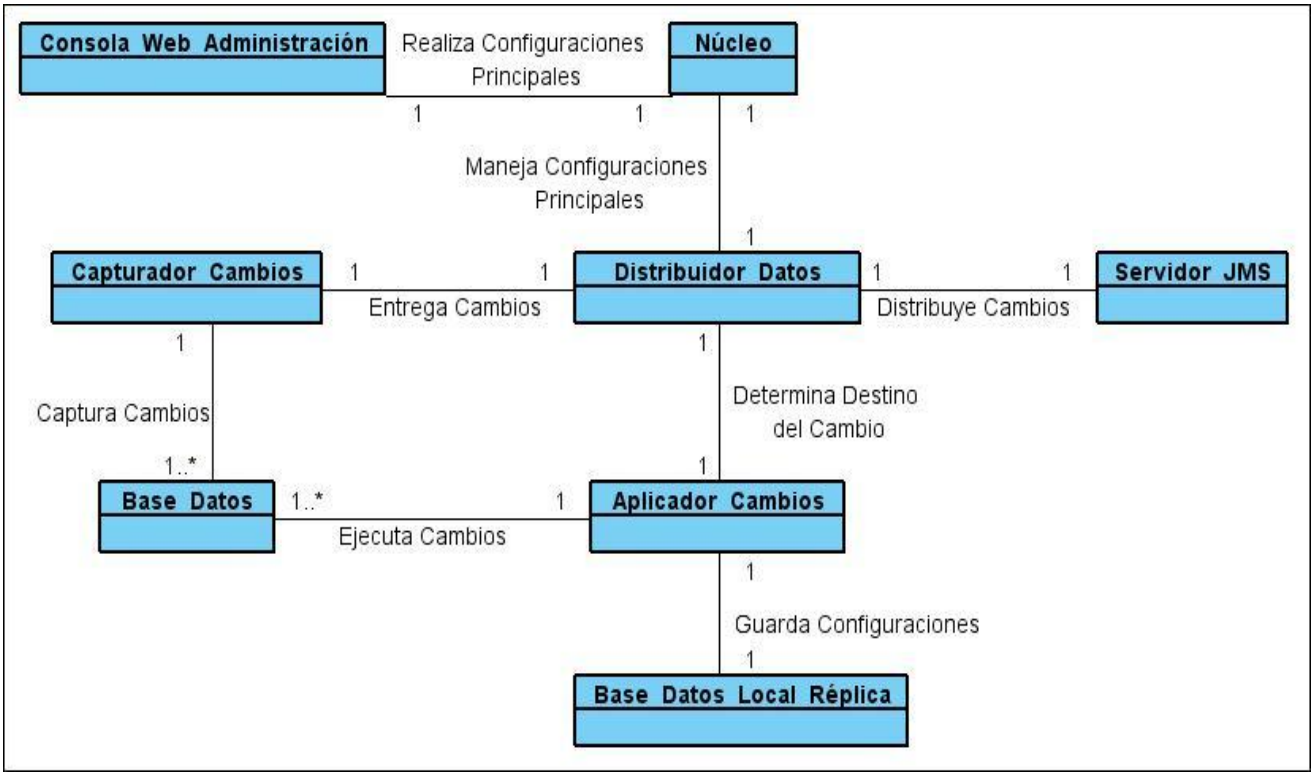


Fig. #1: Diagrama de Clases del Modelo de Dominio.

2.5 Modelado del Sistema

Un requerimiento no es más que la "*condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente*". [15]

2.5.1 Requisitos Funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir; permiten expresar específicamente las responsabilidades del sistema que se propone, determinar de una manera clara lo que el sistema debe hacer y no alteran las funcionalidades del producto.

RF 1: Enviar datos de gran tamaño por FTP.
RF 2: Almacenar archivo binario temporal en servidor FTP.
RF 3: Recibir datos gran tamaño por FTP.
FR 4: Eliminar archivo binario del servidor FTP.
RF 5: Actualizar archivo binario temporal del servidor FTP.

2.5.2 Requisitos No Funcionales

Los requerimientos no funcionales son propiedades o cualidades que el sistema debe tener.

Hardware:

Estaciones de trabajo:

- RAM 128 MB o superior.
- Procesador 800 MHz como mínimo.
- Disco Duro 40 MB o superior.

Servidores:

- RAM 1 GB o superior.
- Procesador 2.0 GHz o superior.
- Disco Duro 180 GB o superior.

Software:

Estaciones de trabajo:

- Internet Explorer v6.0 o superior.
- Mozilla Firefox v2.0 o superior.

Servidores:

- Servidor de mensajería JMS ActiveMQ.
- Servidor FTP que implemente el comando *Resume*.

Legales:

- Las herramientas y componentes empleados en el desarrollo del módulo deben estar basados en Software Libre.

2.5.3 Casos de Uso del Sistema

- CU Enviar archivo binario.
- CU Recibir archivo binario.

2.5.4 Diagrama de Casos de Uso del Sistema

Diagrama de Casos de Uso del Sistema

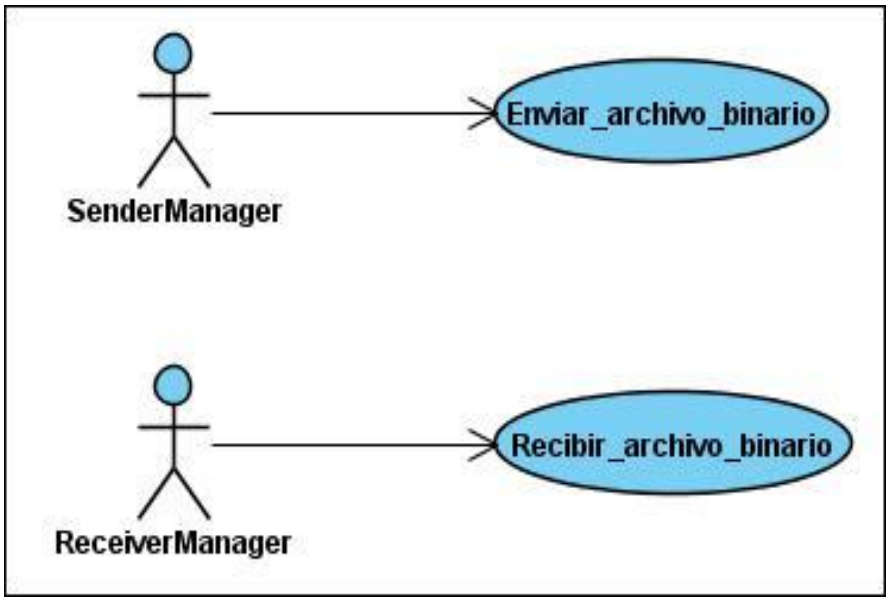


Fig. #2: Diagrama de Casos de Uso del Sistema.

Descripción de los Actores del Sistema

SenderManager: se corresponde con una clase incluida en el componente Distribuidor de Datos y es la responsable de la transmisión de la información.

ReceiverManager: se corresponde con una clase incluida en el componente Distribuidor de Datos y es la responsable de la recepción de la información.

2.5.5 Descripción de Casos de Uso del Sistema

CU Enviar archivo binario

Caso de Uso:	Enviar archivo binario
Actor(es):	Clase SenderManager
Resumen:	El caso de uso se inicia cuando la clase SenderManager comprueba si la transacción posee algún tipo de dato binario (Blob, Clob, Oid, Bytea). Luego toma el dato y lo convierte en fichero, almacenándolo en la carpeta temporal de Reko. Seguidamente el hilo de ejecución lo enviará al servidor FTP.
Precondiciones:	Las instancias de Reko involucradas en la réplica deben de estar conectadas a los servidores JMS ActiveMQ y al servidor FTP; ambos deben de estar en ejecución.
Referencias	RF1, RF2
Prioridad	Crítico

Flujo Normal de Eventos	
Sección "Filtrar Fichero Binario"	
Acción del Actor	**Respuesta del Sistema**
1- La Clase SenderManager recibe una transacción.	
	2- El sistema comprueba la existencia de algún dato binario.
	3- Toma el dato binario y lo convierte en un fichero.
	4- Salva el fichero en la carpeta temporal de Reko.
5- Envía la transacción.	
Sección "Subir Fichero Binario al Servidor FTP"	
Acción del Actor	**Respuesta del Sistema**
1- Se ejecuta toda la sección "Filtrar Fichero Binario".	
2- Inicializa la tarea que se ejecutará	

periódicamente.	3- Comprueba la existencia de ficheros en la carpeta temporal de Reko. 4- Establece conexión con el servidor FTP y envía el fichero. 5- El sistema espera un tiempo determinado y vuelve al paso 3 de la Sección "Subir Fichero al Servidor FTP".
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	3.1 - Si no existen ficheros, espera un tiempo determinado y vuelve al paso 3 de la Sección "Subir Fichero al Servidor FTP".
	4.1 - Si la conexión con el servidor es interrumpida, se restablece y se resume el envío del fichero.
Poscondiciones	Se ha enviado una transacción.

Caso de Uso Recibir Archivo Binario

Caso de Uso:	Recibir archivo binario
Actor(es):	Clase ReceiverManager
Resumen:	El caso de uso se inicia cuando la clase ReceiverManager recibe una transacción y comprueba si posee alguna referencia a un dato binario. Si lo posee añade esa referencia a una lista de descargas. Se ejecuta la descarga de todos los elementos de la lista. El dato binario es añadido a la transacción.
Precondiciones:	- Las instancias de Reko involucradas en la réplica deben de estar conectadas a los servidores JMS ActiveMQ y al servidor FTP; ambos deben estar en ejecución.
Referencias	RF3, RF4, RF5
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- La clase ReceiverManager recibe una transacción. 7- Procesa la transacción.	2- El sistema verifica si posee una referencia a un tipo de dato binario. 3- Toma la referencia y la añade a una lista de descargas. 4- Chequea el contenido de la carpeta principal del servidor FTP. 5- Descarga el fichero existente en la lista de descargas. 6- Luego de descargado, añade el dato a la transacción.
Poscondiciones	Se ha recibido una transacción.

2.6 Propuesta del Sistema

El módulo a implementar estará compuesto por dos partes fundamentales: una encargada del envío y otra del recibo de datos de gran tamaño.

Al recibirse una petición de réplica de un campo con datos de gran tamaño, el dato será convertido en un fichero y almacenado en la carpeta temporal de Reko, guardándose su referencia en la transacción. Existirá un hilo de ejecución que se ejecutará periódicamente y será el encargado de enviar estos ficheros temporales al servidor File Transfer Protocol (FTP). Durante el recibo se procesará la transacción y si posee datos de gran tamaño se añadirá su referencia a una lista de descargas. Se realizará la descarga del fichero y será añadido a la transacción, luego de esto se enviará una confirmación: “Fichero recibido”, al nodo que generó la transacción; este nodo verificará que todos los destinatarios hayan descargado el fichero y lo eliminará del servidor FTP.

2.7 Conclusiones

En este capítulo se hizo una descripción del proceso a automatizar así como del funcionamiento interno del software Reko, con el objetivo de lograr un mayor entendimiento del negocio. Posteriormente se realizó el Modelo de Dominio del Negocio, pues no existen procesos observables; siendo definidas como clases cada una de las partes.

Luego de conocer el funcionamiento del negocio, se logró la identificación de requisitos, tanto funcionales como no funcionales. Los Requerimientos Funcionales fueron agrupados por Casos de Uso del Sistema (CUS), quedando definidos dos: Enviar archivo binario y Recibir archivo binario. Estos CUS fueron descritos para obtener una descripción detallada de los mismos, permitiendo, finalmente, realizar una propuesta del funcionamiento de módulo que luego se integrará al sistema.

CAPÍTULO 3: DISEÑO DEL SISTEMA

3.1 Introducción

En el presente capítulo de realiza un estudio del Diseño del módulo; se presenta la Realización de los Casos de Uso, mediante los Diagramas de Clases del Diseño y de Interacción correspondientes a cada uno de ellos. En el trascurso de la representación de los diagramas, se hace alusión a las técnicas empleadas para obtener mejores resultados en la solución, específicamente a los Patrones de Diseño utilizados para la obtención de las Clases del Diseño y la estructuración de los diagramas, además en el capítulo se describe el Diagrama de Despliegue correspondiente al sistema Reko.

3.2 Descripción de la Arquitectura del Software de Réplica Reko

La arquitectura de software está definida por la IEEE 1471-2000 como: *"La organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución."*

Partiendo de esta definición, se concluye que cada sistema de software contará con su propia arquitectura, cuya estructura dependerá de las características específicas de cada proyecto. Inicialmente esta se define como candidata en la Fase de Inicio, pasando luego por un refinamiento en la Fase de Elaboración donde se define como Línea Base de la Arquitectura.

La arquitectura está dada por seis categorías diferentes de Estilos Arquitectónicos, los cuales versan en dependencia de los requerimientos del software. Una de estas categorías está conformada por los Estilos Arquitectónicos de Llamada y Retorno:

- Model-View-Controller (MVC).
- Arquitecturas en Capas.
- Arquitecturas Orientadas a Objetos.
- Arquitecturas Basadas en Componentes.

Descripción del Estilo Arquitectónico Empleado

El desarrollo basado en componentes permite alcanzar un mayor nivel de reutilización de software y que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados, también simplifica el mantenimiento del sistema, entre otras

cuestiones. Un componente típicamente es una "*unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio*". [16]

Partiendo de las definiciones antes expuestas, se puede describir en términos generales a la arquitectura de Reko como Basada en Componentes, pues sus partes encapsulan un conjunto de comportamientos que pueden ser reemplazados por otros.

Los principales componentes presentes en el software son:

- **Capturador de cambios:** Encargado de capturar los cambios que se realizan en la base de datos y entregarlos al distribuidor.
- **Distribuidor:** Determina el destino de cada cambio realizado en la base de datos, los envía y se responsabiliza de su llegada.
- **Aplicador:** Ejecuta en la base de datos los cambios que son enviados hacia él desde otro nodo de réplica.
- **Administración:** Permite realizar las configuraciones principales del software como el registro de nodos, configuración de las tablas a replicar y el monitoreo del funcionamiento.

Independientemente de lo antes expuesto, el componente **Administración** responde a un modelo multicapas, donde cada capa tiene funcionalidades y objetivos precisos, así su implementación se encuentra desacoplada de la programación de cualquier otra y la comunicación con una capa inferior ocurre a través de interfaces. Además de estar separadas lógica y estructuralmente, las capas se encuentran separadas de manera física.

Principales Componentes de Reko

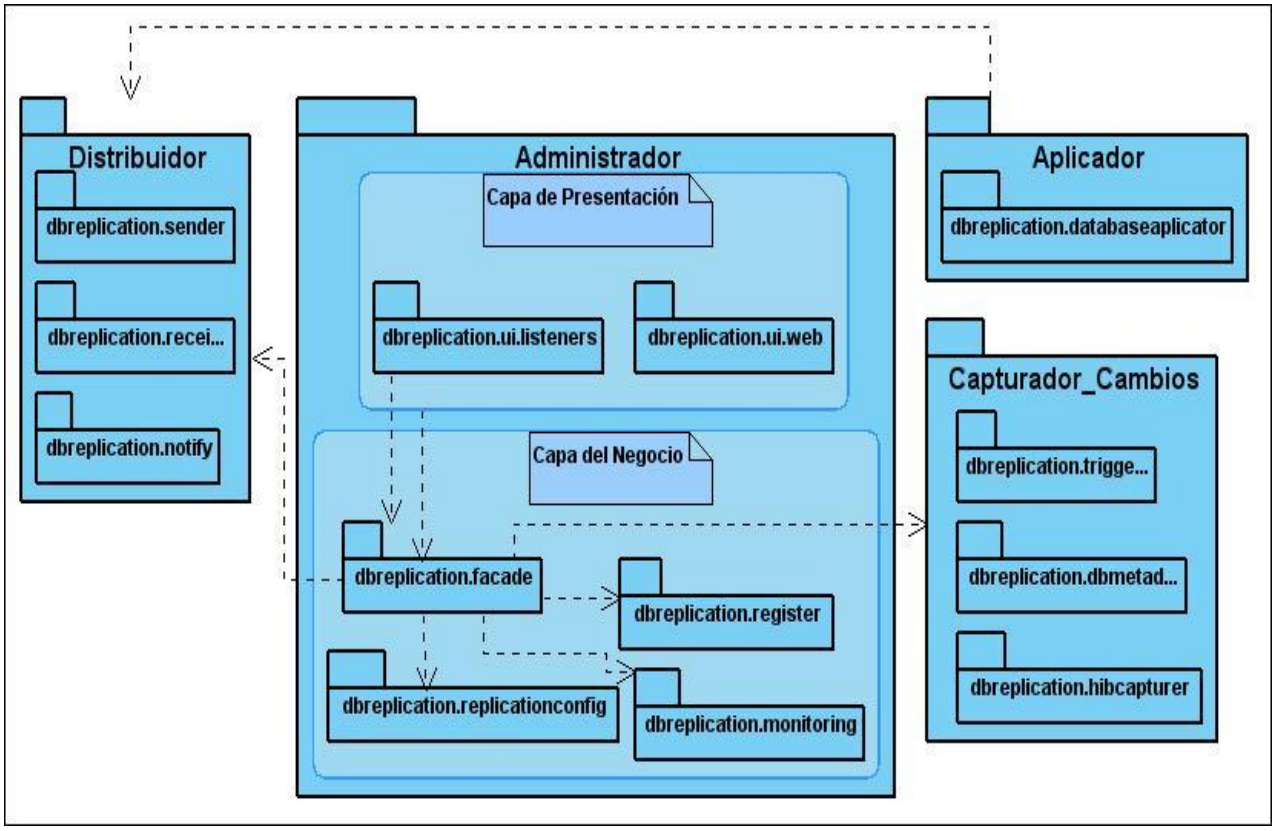


Fig. #3: Vista de los Principales Componentes de Reko, agrupados por paquetes.

Para la trasmisión de datos de gran tamaño, se hace necesaria una integración total con el componente **Distribuidor de Datos**; en el paquete Triggers, correspondiente al componente **Capturador_Cambios** se realizarán modificaciones, pues, hasta el momento, su funcionamiento se basa en cargar estos datos en la memoria RAM y luego son transferidos; lo cual es práctico para datos pequeños, pero al enviar datos de gran tamaño, ocurriría una sobrecarga. Con este mismo fin será modificado el componente **Aplicador.**

Patrones de Diseño:

Los patrones de diseño son *"la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces."* Un patrón de diseño es una solución a un problema de diseño; facilitan la reusabilidad, extensibilidad y mantenimiento. [17]

Para dar solución al problema planteado se hizo uso de estos patrones, que se clasifican en patrones de diseño GRASP y GOF.

Patrones GRASP: Los Patrones Generales de Software para Asignar Responsabilidades (General Responsibility Assignment Software Patterns) describen los principios fundamentales de asignación de responsabilidades a objetos, expresado en formas de patrones. [18]

- **Experto:** Genera las clases para la gestión de las entidades con las responsabilidades debidamente asignadas, pues cada una de estas clases cuenta con un conjunto de funcionalidades relacionadas directamente con la entidad que representan.
- **Creador:** Guía la asignación de responsabilidades relacionadas con la creación de objetos. Su propósito fundamental es encontrar un creador que debe ser conectado con el objeto producido en cualquier evento, lo cual da soporte al bajo acoplamiento.
- **Alta Cohesión:** La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase; por lo que puede ser calificada como alta cuando existen las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Este patrón se basa en la asignación de responsabilidades, de modo que la cohesión siga siendo alta.

Patrones GOF:

- **Cadena de responsabilidad (Chain of Responsibility):** Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada. Puede ser implementada al crear una clase “manipuladora” (handler) abstracta (para especificar la interfaz y funcionalidad genérica) y crear subclases concretas para definir varias posibles implementaciones. Sin embargo, no existen requerimientos absolutos para todos los miembros de la cadena de responsabilidad, por descender a partir de la misma clase, proporcionando que todos ellos soporten la interfaz necesaria para integrar con otros miembros de la cadena.
- **Orden (Command):** Es un patrón de diseño de comportamientos que encapsula una operación en un objeto. Permite ejecutar operaciones como crear una cola de objetos, gestionarla y poder deshacer operaciones sin necesidad de conocer el contenido de la misma.

3.3 Modelo de Diseño

El Modelo de Diseño es un modelo de objetos que describe la realización de los CUS y sirve como abstracción del Modelo de Implementación y del Código Fuente. Es utilizado como entrada fundamental para las actividades de los Flujos de Trabajo de Implementación y Pruebas. Puede ser calificado como un artefacto integral pues abarca todas las clases del diseño, subsistemas, paquetes, colaboraciones y las relaciones existentes entre ellos.

3.3.1 Realización de Casos de Uso del Diseño

Para la realización de los Casos de Uso del Diseño, es necesario hacer uso de dos tipos de diagramas: Estáticos y Dinámicos, los cuales están dados por los Diagramas de Clases del Diseño y los Diagramas de Interacción, respectivamente.

Diagramas de Clases del Diseño

Un Diagrama de Clases del Diseño representa concretamente lo que debe ser implementado, siendo la representación estática del sistema, a través de las clases y sus relaciones.

Diagrama de Clases del Diseño. CU Enviar Archivo Binario

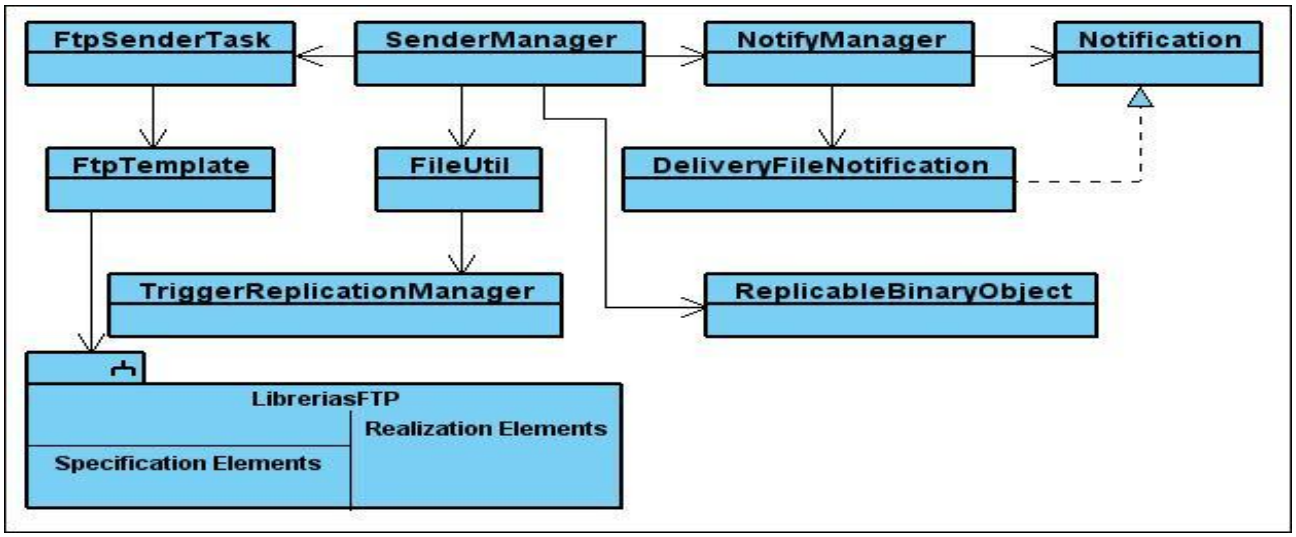


Fig. #4: Diagrama de clases del diseño. CU Enviar archivo binario.

Diagrama de Clases del Diseño. CU Recibir Archivo Binario

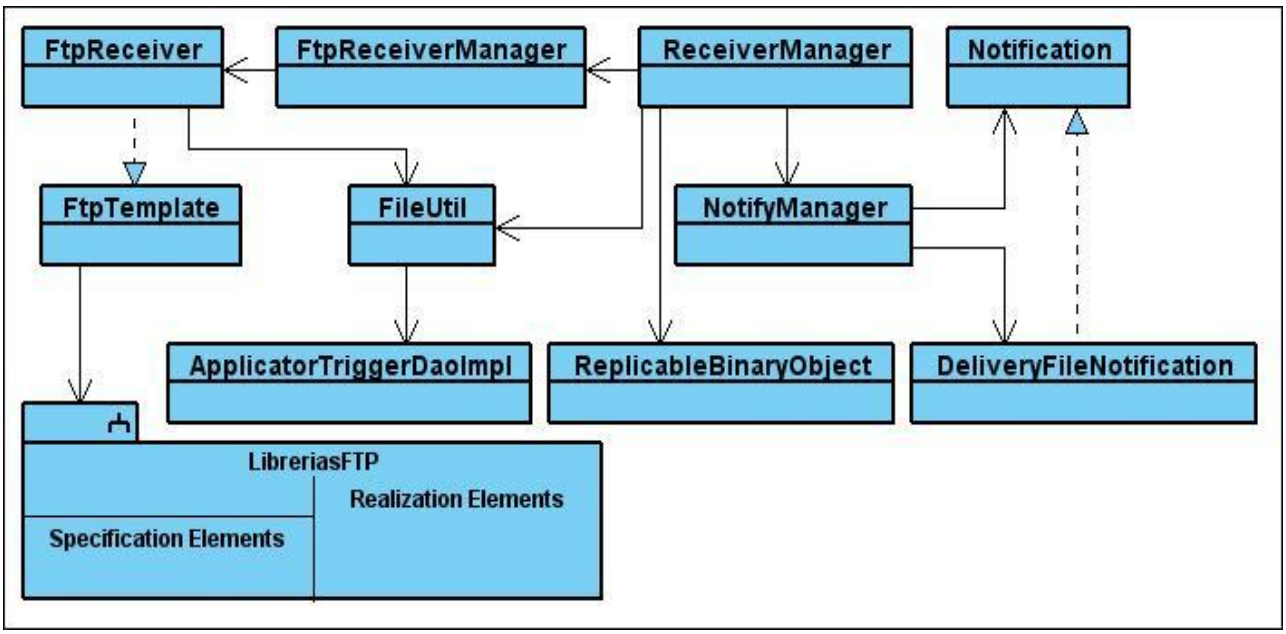


Fig. #5: Diagrama de clases del diseño. CU Recibir archivo binario.

Descripción de las clases

Descripción de la Clase SenderManager	
Nombre: SenderManager	
Tipo de clase: Controladora	
Atributo:	**Tipo:**
Logger	protected
Sending	private
Sender	private
FtpSenderManager	private
Transactions	private
Waiting	private
UseFtp	private

FileWriter	private
Sendtask	private
NumberOfTargets	private
Timer	private
timer1	private
ConvertToFile	private
FileUtil	private

Responsabilidades	
Nombre:	addTransaction (transaction : cu.uci.dbreplication.transportable.ReplicableGroup)
Descripción:	Añade la transacción especificada por parámetros a la lista de transacciones.
Nombre:	monitorSend(group : cu.uci.dbreplication.transportable.ReplicableGroup, targets : java.util.Set<String>)
Descripción:	Envía transacciones al objetivo especificado.
Nombre:	sendIfAplicable(transaction : cu.uci.dbreplication.transportable.ReplicableGroup)
Descripción:	Aplica configuraciones a la transacción y la envía a partir del método monitorSend.
Nombre:	sendComplete(transaction : cu.uci.dbreplication.transportable.ReplicableGroup)
Descripción:	Envía la transacción.
Nombre:	send(group : cu.uci.dbreplication.transportable.ReplicableGroup)
Descripción:	Comprueba si se está enviando, de lo contrario envía para cada uno de los objetivos.
Nombre:	send(group : cu.uci.dbreplication.transportable.ReplicableGroup, targetId : String)
Descripción:	Construye el ReplicableGroup.
Nombre:	setSender(sender : cu.uci.dbreplication.sender.Sender)
Descripción:	Permite modificar el objeto Sender.
Nombre:	applyFilter(transaction : cu.uci.dbreplication.transportable.ReplicableGroup)
Descripción:	Aplica los filtros de destinatario al ReplicableGroup.
Nombre:	convertBlobs(transaction : cu.uci.dbreplication.transportable.ReplicableGroup)

Descripción:	Extrae el dato binario y lo reemplaza por la referencia a él.
Nombre:	generateFileName(action : int, groupId : String, paramIndex : int)
Descripción:	Genera el nombre del fichero que se salvará temporalmente.
Nombre:	sendTransactions()
Descripción:	Mientras existen transacciones las envía.
Nombre:	setNotifyManager(notifyManager : cu.uci.dbreplication.notify.NotifyManager)
Descripción:	Permite modificar la instancia de la clase NotifyManager.
Nombre:	setFileUtil(fileUtil : cu.uci.dbreplication.utils.file.FileUtil)
Descripción:	Permite modificar la instancia de FileUtil.
Nombre:	afterPropertiesSet()
Descripción:	Método a implementar de la interfaz InitializingBean. Inicia las tareas de ejecución periódica.

Descripción de la clase ReceiverManager	
Nombre: ReceiverManager	
Tipo de clase: Controladora	
Atributo:	**Tipo:**
Logger	protected
FtpReceiverManager	private
FileReader	private
ImportingDB	private
Fileut	public
BinaryList	private
Responsabilidades:	
Nombre:	onTransactionArrive(transaction : cu.uci.dbreplication.transportable.ReplicableGroup)
Descripción:	Método que recibe la transacción enviada por otros nodos de réplica.
Nombre:	persistInLocalDB(group : cu.uci.dbreplication.transportable.ReplicableGroup)

Descripción:	Antes de aplicar la transacción, el grupo es salvado localmente en la base de datos Berkely.
Nombre:	downloadBinaryDataIfAplicable(group:cu.uci.dbreplication.transportable.ReplicableGroup)
Descripción:	Descarga el fichero binario, de poseer la transacción una referencia a este.
Nombre:	isDataComplete(group : cu.uci.dbreplication.transportable.ReplicableGroup)
Descripción:	Comprueba que esté completa la transacción antes de aplicarla.
Nombre:	afterPropertiesSet()
Descripción:	Método a implementar de la interfaz InitializingBean.
Nombre:	setFtpReceiverManager(ftpReceiverManager : cu.uci.dbreplication.receiver.ftp.FtpReceiverManager)
Descripción:	Permite modificar la instancia de la clase FtpReceiverManager.

Descripción de la Clase FTPSenderTask	
Nombre: FTPSenderTask	
Tipo de Clase: Auxiliar	
Atributo:	**Tipo:**
FtpSenderManager	private
Responsabilidades:	
Nombre:	run()
Descripción:	Método heredado de TimerTask que contiene la tarea a ejecutar periódicamente.
Nombre:	setFtpSenderManager(ftpSenderManager : cu.uci.dbreplication.sender.ftp.FtpSenderManager)
Descripción:	Permite modificar la instancia de la clase FtpSenderManager.

Descripción de la Clase FTPTemplate	
Nombre: FTPTemplate	
Tipo de clase: Auxiliar	
Atributo:	**Tipo:**
Logger	protected
transfersPrefix	protected
Host	private
Port	private
username	private
password	private
initialDir	private
dataDir	Private
Log	Protected
ftp	protected
eventPublisher	protected
Responsabilidades:	
Nombre:	getHost()
Descripción:	Obtiene el servidor.
Nombre:	FtpTemplate()
Descripción:	Constructor de la clase.
Nombre:	init()
Descripción:	Configura los parámetros de conexión y conecta al servidor FTP.
Nombre:	connect()
Descripción:	Inicia la conexión con el servidor FTP.
Nombre:	disconnect()
Descripción:	Finaliza la conexión con el servidor FTP.

Nombre:	afterPropertiesSet()
Descripción:	Método a implementar de la interfaz InitializingBean.
Nombre:	setHost(host : String)
Descripción:	Permite modificar el servidor FTP.
Nombre:	setUsername(username : String)
Descripción:	Permite modificar el nombre de usuario.
Nombre:	setPassword(password : String)
Descripción:	Permite modificar la contraseña de acceso.
Nombre:	getPort()
Descripción:	Obtiene el puerto de conexión.
Nombre:	setPort(port : int)
Descripción:	Permite modificar el puerto de conexión.
Nombre:	getInitialDir()
Descripción:	Obtiene la dirección inicial en el servidor FTP.
Nombre:	setInitialDir(initialDir : String)
Descripción:	Permite modificar la dirección inicial del servidor FTP.
Nombre:	getDataDir()
Descripción:	Obtiene la dirección del fichero temporal de Reko.
Nombre:	setDataDir(dataDir : String)
Descripción:	Permite modificar la dirección del fichero temporal de Reko.
Nombre:	uploadFile(fichlocal : String, nombrefichremoto : String)
Descripción:	Subir fichero desde la carpeta temporal de Reko al directorio inicial del servidor FTP.
Nombre:	descargarFichero(ficheroLocal : String, ficheroRemoto : String)
Descripción:	Permite descargar el fichero desde el servidor FTP.
Nombre:	renameArchive(oldname : String, newname : String)
Descripción:	Permite cambiar el nombre del fichero en el servidor FTP.

Nombre:	removeRemoteFile(filename : String)
Descripción:	Permite eliminar el fichero del servidor FTP.

Descripción de la clase FileUtil	
Nombre: FileUtil	
Tipo de clase: Auxiliar	
Atributo:	**Tipo:**
Path	private
Logger	protected
BUFFER_SIZE	private
Responsabilidades:	
Nombre:	FileUtil()
Descripción:	Constructor de la clase.
Nombre:	saveFile(fileName : String, io : java.io.InputStream)
Descripción:	Permite salvar el fichero en la carpeta temporal de Reko.
Nombre:	removeFile(fileName : String)
Descripción:	Permite eliminar el fichero de la carpeta temporal de Reko.
Nombre:	setBUFFER_SIZE(buffer_size : int)
Descripción:	Ajusta el tamaño del buffer.
Nombre:	getInitialPath()
Descripción:	Obtiene la dirección de la carpeta temporal de Reko.
Nombre:	setInitialPath(initialPath : String)
Descripción:	Permite modificar la dirección de la carpeta temporal de Reko.
Nombre:	renameFile(oldName : String, newName : String)
Descripción:	Permite cambiar el nombre al archivo dentro de la carpeta temporal de Reko.
Nombre:	readFile(fileName : String)

Descripción:	Permite leer el archivo dentro de la carpeta temporal de Reko.
Nombre:	readFiletoByteArray(fileName : String)
Descripción:	Convierte el fichero leído a un arreglo de bytes.

Descripción de la clase NotifyManager	
Nombre: NotifyManager	
Tipo de clase: Auxiliar	
Atributo:	**Tipo:**
Logger	protected
senderManager	private
deliverySender	private
receiverManager	private
Responsabilidades:	
Nombre:	onNotificationReceived(notification : cu.uci.dbreplication.notify.Notification)
Descripción:	Separa las notificaciones de acuerdo a su tipo.
Nombre:	onDeliveryFileNotification(notification : cu.uci.dbreplication.notify.DeliveryFileNotification)
Descripción:	Notifica al SenderManager que un nodo ha recibido un fichero.
Nombre:	setSenderManager(senderManager : cu.uci.dbreplication.sender.SenderManager)
Descripción:	Permite modificar la instancia de la clase SenderManager.
Nombre:	setReceiverManager(receiverManager : cu.uci.dbreplication.receiver.ReceiverManager)
Descripción:	Permite modificar la instancia de la clase ReceiverManager.
Nombre:	sendNotification(notification : cu.uci.dbreplication.notify.Notification)
Descripción:	Envía una notificación.
Nombre:	afterPropertiesSet()

Descripción:	Método a implementar de la interfaz InitializingBean.

Descripción de la clase DeliveryFileNotification	
Nombre: DeliveryFileNotification	
Tipo de clase: Auxiliar	
Atributo:	**Tipo:**
Filename	public
Filename	private
Responsabilidades:	
Nombre:	DeliveryFileNotification(status : int, target : String, toTarget : String, transactionId : String, afilename : String)
Descripción:	Constructor de la clase.
Nombre:	getFileName()
Descripción:	Obtiene el nombre del archivo.
Nombre:	setFileName(fileName : String)
Descripción:	Permite cambiar el nombre del archivo.

Descripción de la clase Notification	
Nombre: Notification	
Tipo de clase: Auxiliar	
Atributo:	**Tipo:**
Target	private
transactionId	private
ToTarget	private
Status	private
JUST_RECEIVED	public

EXECUTE_SUCCESS	public
EXECUTE_CONFLICT	public
Filename	private
Responsabilidades:	
Nombre:	getTarget()
Descripción:	Obtiene el objetivo.
Nombre:	setTarget(target : String)
Descripción:	Permite modificar el objetivo.
Nombre:	getTransactionId()
Descripción:	Obtiene el identificador de la transacción.
Nombre:	setTransactionId(transactionId : String)
Descripción:	Permite modificar el identificador de la transacción.
Nombre:	getStatus()
Descripción:	Obtiene el estado de la transacción.
Nombre:	setStatus(status : int)
Descripción:	Permite modificar el estado de la transacción.
Nombre:	getToTarget()
Descripción:	Obtiene el destino al que es enviada la notificación.
Nombre:	setToTarget(toTarget : String)
Descripción:	Permite modificar el destino al que será enviada la notificación.
Nombre:	Notification(status : int, target : String, toTarget : String, transactionId : String)
Descripción:	Constructor de la clase.
Nombre:	Notification()
Descripción:	Constructor de la clase.
Nombre:	getFilename()
Descripción:	Obtiene el nombre del archivo del cual es enviada la notificación.

Nombre:	setFilename(filename : String)
Descripción:	Permite cambiar el nombre al archivo del cual se enviará la notificación.

Descripción de la clase ReplicableBinaryObject	
Nombre: ReplicableBinaryObject	
Tipo de clase: Auxiliar	
Atributo:	**Tipo:**
BLOB	public
groupAction	private
binaryType	private
downloaded	private
Responsabilidades:	
Nombre:	ReplicableBinaryObject(binaryType : int, downloaded : boolean, groupAction : cu.uci.dbreplication.receiver.ftp.GroupAction)
Descripción:	Constructor de la clase.
Nombre:	getBinaryType()
Descripción:	Obtiene el tipo binario.
Nombre:	setBinaryType(binaryType : int)
Descripción:	Permite modificar el tipo binario.
Nombre:	isDownloaded()
Descripción:	Devuelve True, si se ha descargado y False en caso contrario.
Nombre:	setDownloaded(downloaded : boolean)
Descripción:	Permite cambiar el estado de descarga.
Nombre:	getGroupAction()
Descripción:	Devuelve el grupo sobre el cual se realizarán las operaciones.

Descripción de la clase TriggerReplicationManager	
Nombre: TriggerReplicationManager	
Tipo de clase: Controladora	
Responsabilidades:	
Nombre:	getReplicableActionInsert
Descripción:	Obtiene las acciones a replicar de tipo “Insert”.
Nombre:	getReplicableActionUpdate
Descripción:	Obtiene las acciones a replicar de tipo “Update”.
Nombre:	getReplicableActionDelete
Descripción:	Obtiene las acciones a replicar de tipo “Delete”.

Descripción de la clase ApplicatorTriggerDaoImpl	
Nombre: ApplicatorTriggerDaoImpl	
Tipo de clase: Controladora	
Responsabilidades:	
Nombre:	Execute
Descripción:	Aplica los cambios en la base de datos a replicar.

Diagramas de Interacción del Diseño. Secuencia

Los Diagramas de Interacción muestran las interacciones entre objetos mediante transferencia de mensajes entre objetos o subsistemas, por lo que son empleados para modelar los aspectos dinámicos del sistema. Estos diagramas pueden ser clasificados como Diagramas de Secuencia y Diagramas de Colaboración, los cuales son semánticamente equivalentes, pero sin embargo los de Secuencia destacan el orden temporal de los mensajes y los de Colaboración destacan el orden estructural de los objetos que interactúan.

Para la presente investigación se elaborarán Diagramas de Secuencia, pues la principal intención es mostrar el orden temporal de los mensajes existentes entre las clases del sistema.

Diagrama de Secuencia. CU Enviar Archivo Binario. Sección "Filtrar Fichero Binario"

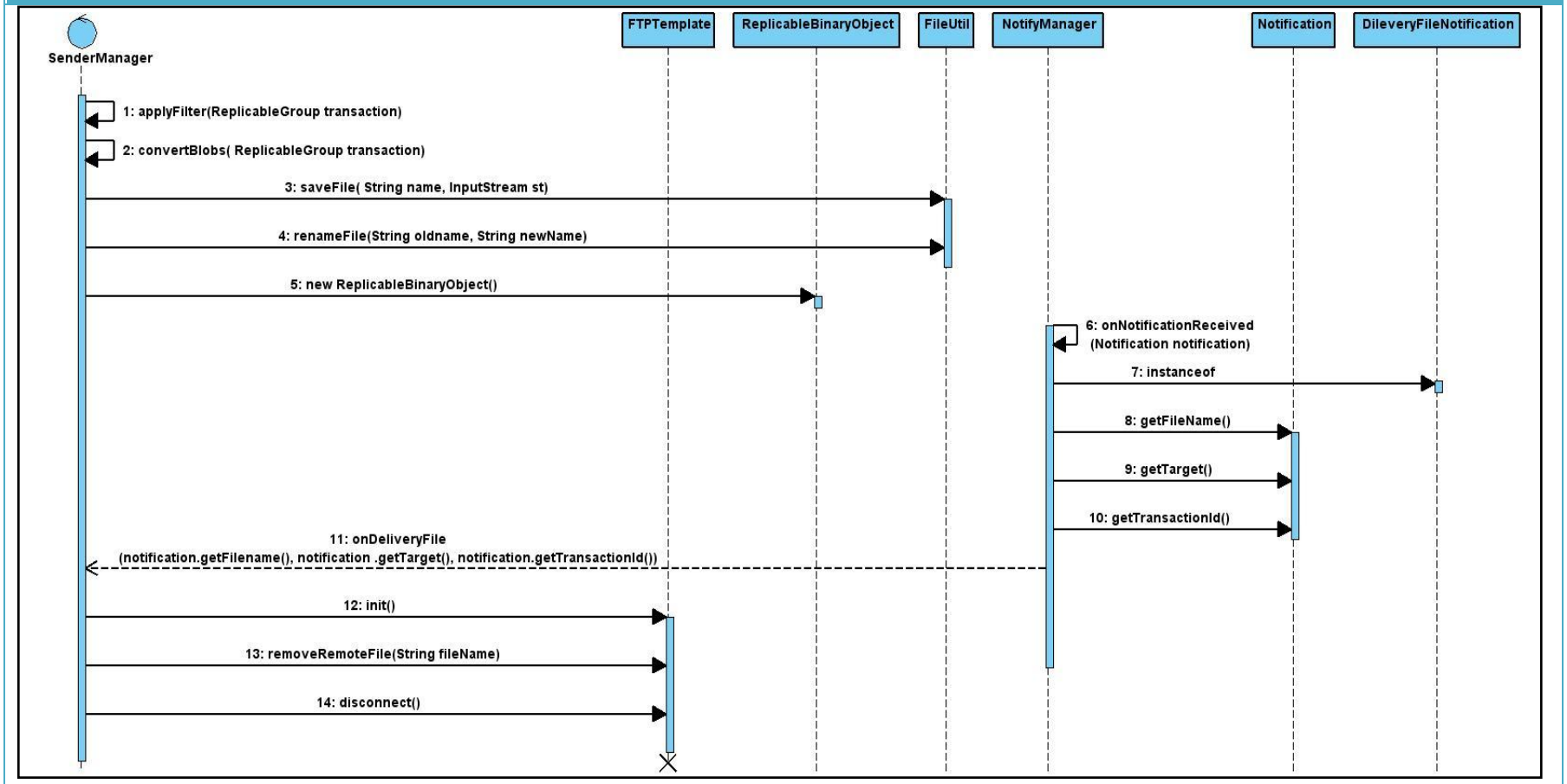


Fig. #6: Diagrama de Secuencia. CU Enviar archivo binario. Sección "Filtrar Fichero Binario".

Diagrama de Secuencia. CU Enviar Archivo Binario. Sección "Subir Fichero Binario"

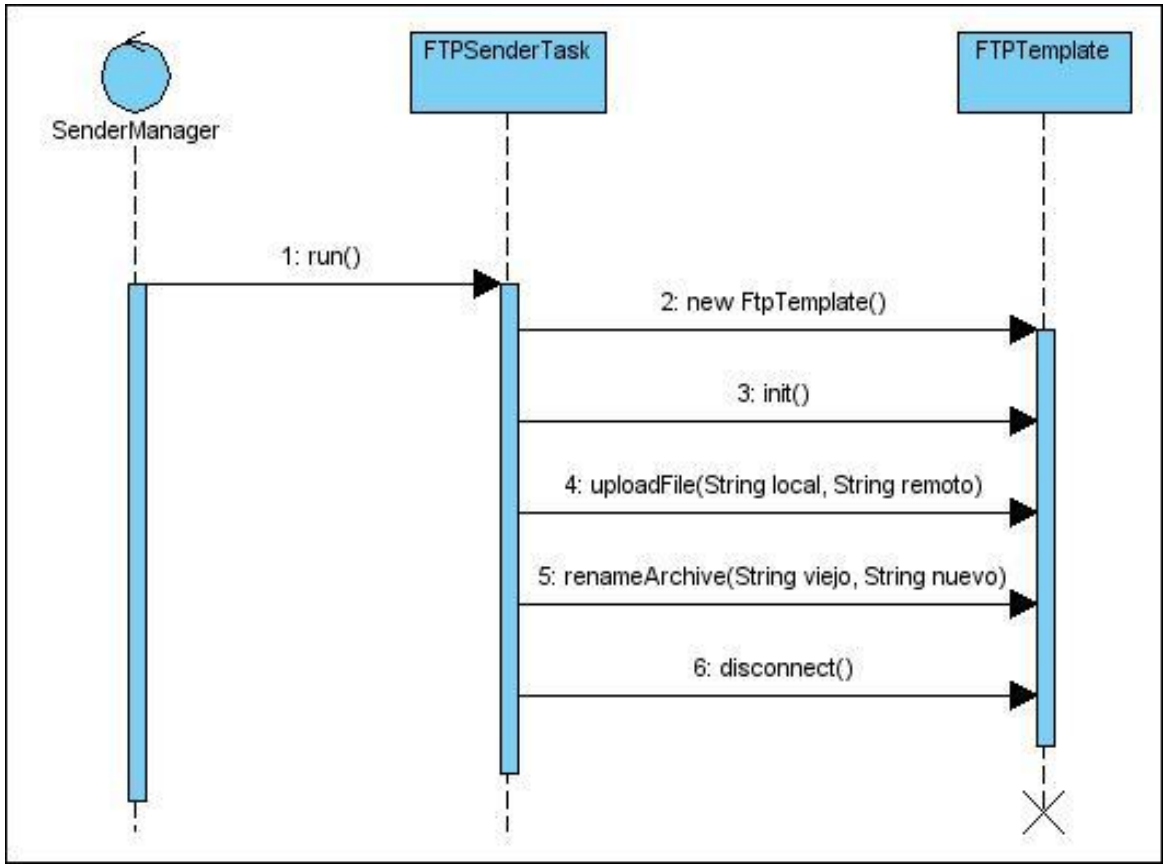


Fig. #7: Diagrama de Secuencia. CU Enviar archivo binario. Sección "Subir Fichero Binario".

Diagrama de Secuencia. CU Recibir Archivo Binario

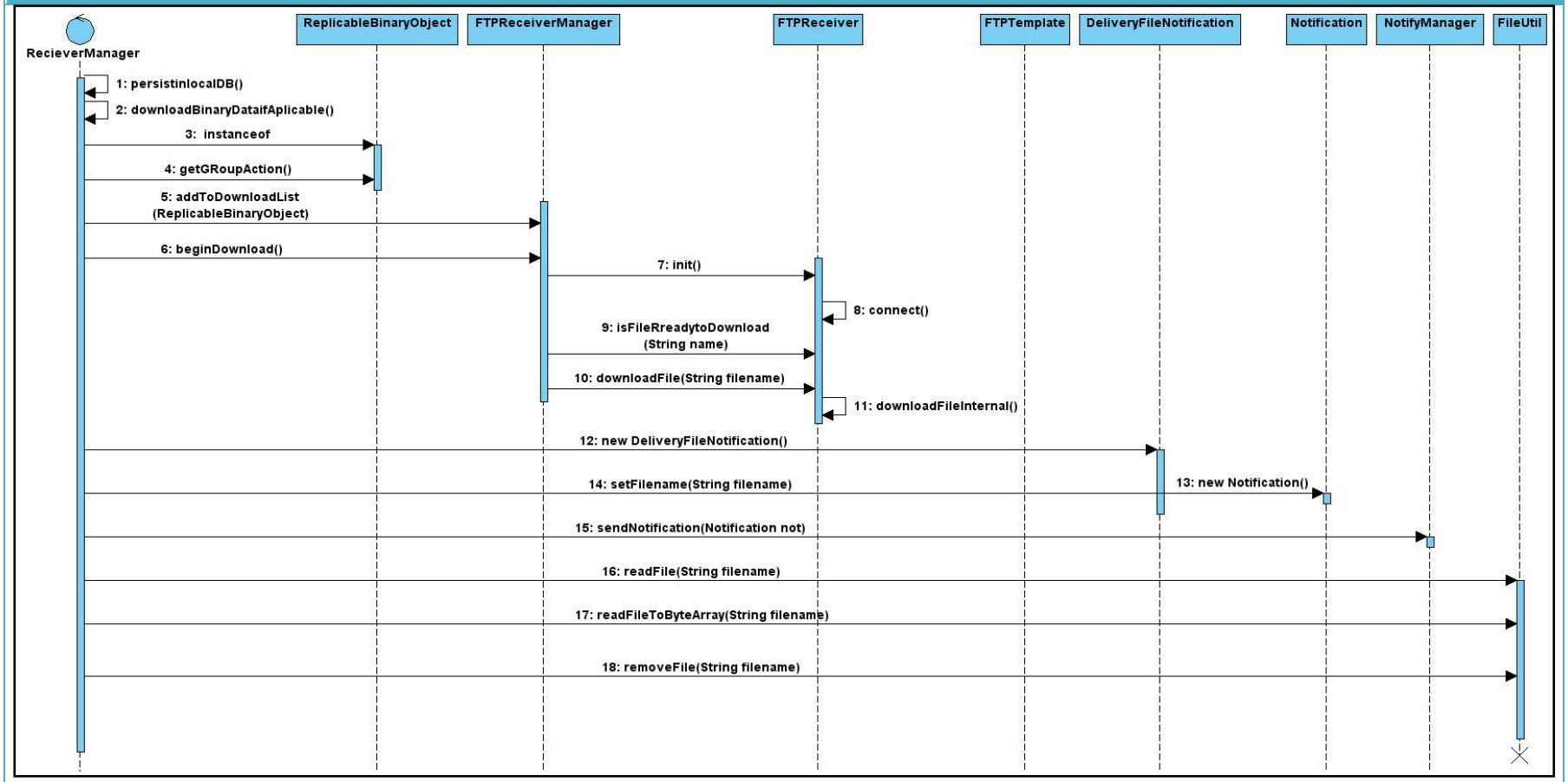


Fig. #8: Diagrama de Secuencia. CU Recibir archivo binario.

3.4 Modelo de Despliegue

El Modelo de Despliegue *"muestra la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos"* [19]

La Vista de Despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación.

Modelo de Despliegue. Reko

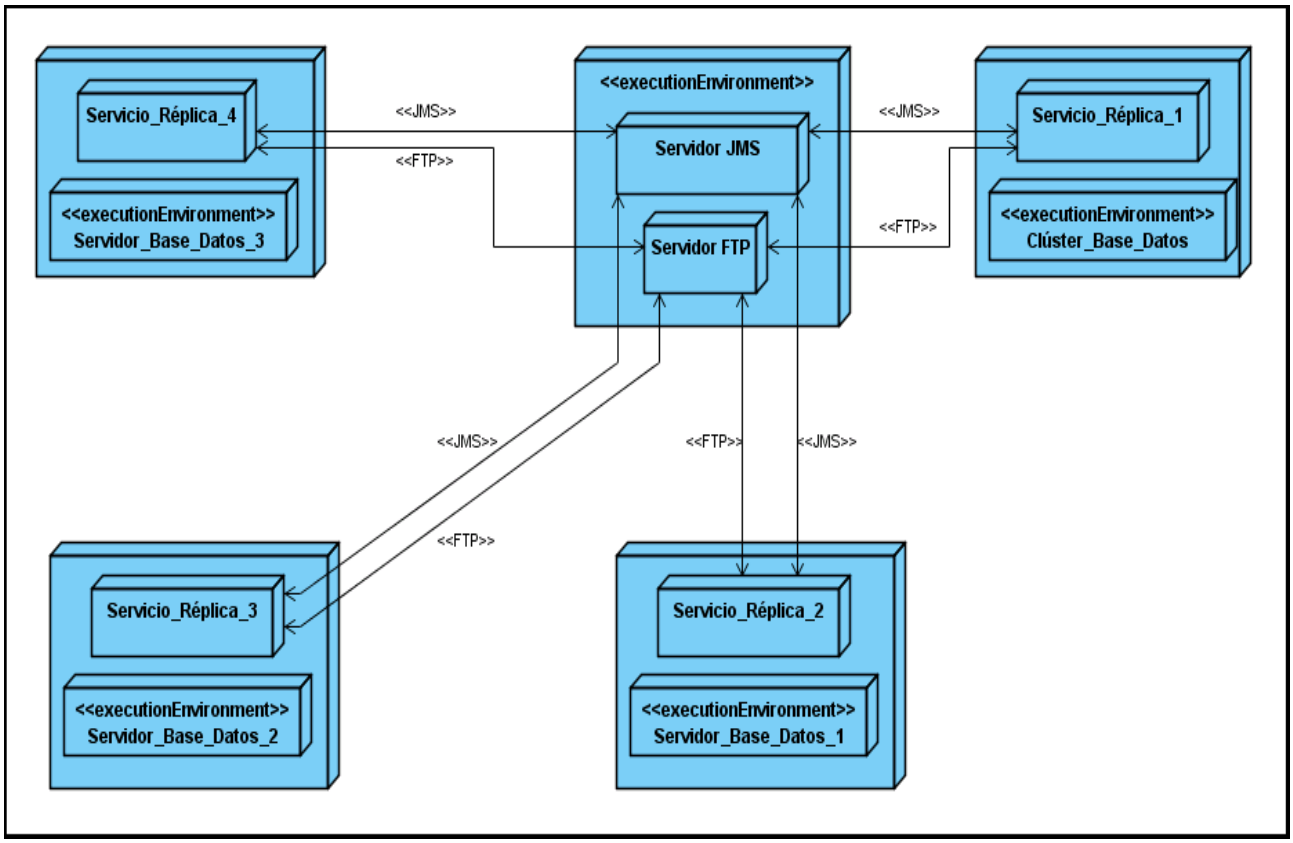


Fig. #9: Modelo de Despliegue. Reko

3.5 Conclusiones

En este capítulo se realizó una Descripción de la Arquitectura del software de réplica Reko, donde se explicó detalladamente cada parte de su estructura. Se describió el Estilo Arquitectónico empleado para su desarrollo: Estilo de Llamada y Retorno, Basado en Componentes; así como los Patrones de Diseño utilizados para dar solución a la problemática planteada.

Se presentaron los Diagramas de Clases del Diseño, correspondientes a los CUS Críticos: Enviar archivo binario y Recibir archivo binario, donde cada una de las clases utilizadas fueron brevemente descritas; se construyeron los Diagramas de Interacción correspondientes a estos Diagramas de Clases del Diseño (específicamente Diagramas de Secuencia).

Y por último se elaboró y expuso el Diagrama de Despliegue correspondiente a Reko como software de réplica, lo cual facilitó el entendimiento de cómo estará físicamente distribuido el sistema.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBAS

4.1 Introducción

En el capítulo correspondiente a Implementación y Pruebas, se expondrá el Diagrama de Componentes, donde se describen los elementos físicos del sistema y sus relaciones. Se muestran las clases e implementaciones más importantes para el módulo y las pruebas realizadas para validar el mismo.

4.2 Modelo de Implementación

El Modelo de Implementación describe cómo los elementos del Modelo de Diseño, se implementan en términos de Componentes, Ficheros de Código Fuente, Ejecutables, etc. Describe también cómo se organizan los Componentes de acuerdo a los mecanismos de estructuración y modularización disponibles en el entorno de implementación y lenguaje o lenguajes de implementación empleados, y cómo dependen los Componentes unos de otros. [20]

4.2.1 Diagrama de Componentes

Un componente es el empaquetamiento físico de los elementos de un modelo, como son las clases en el Modelo de Diseño. *"Los diagramas de componentes describen los elementos físicos y sus realizaciones en el entorno"* de implementación y en el lenguaje o lenguajes de programación utilizados, y cómo dependen los componentes unos de otros. [21]

Diagrama de Componentes. CU Enviar archivo binario.

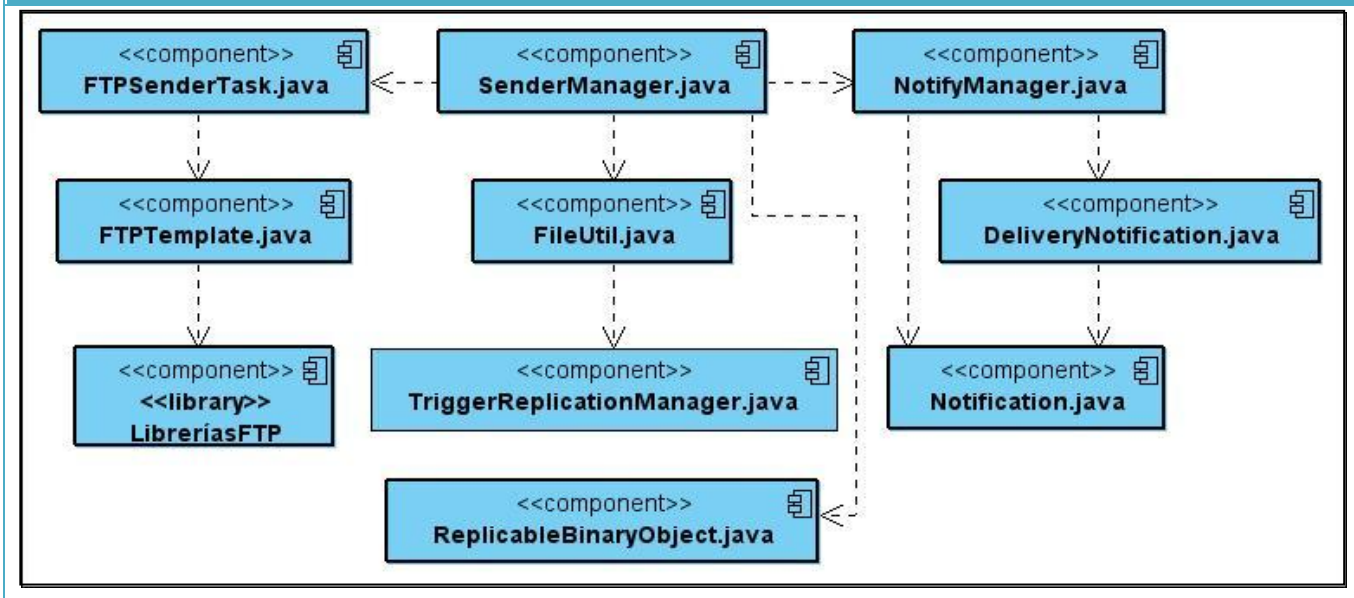


Fig. #10: Diagrama de Componentes. CU Enviar archivo binario.

Diagrama de Componentes. CU Recibir archivo binario

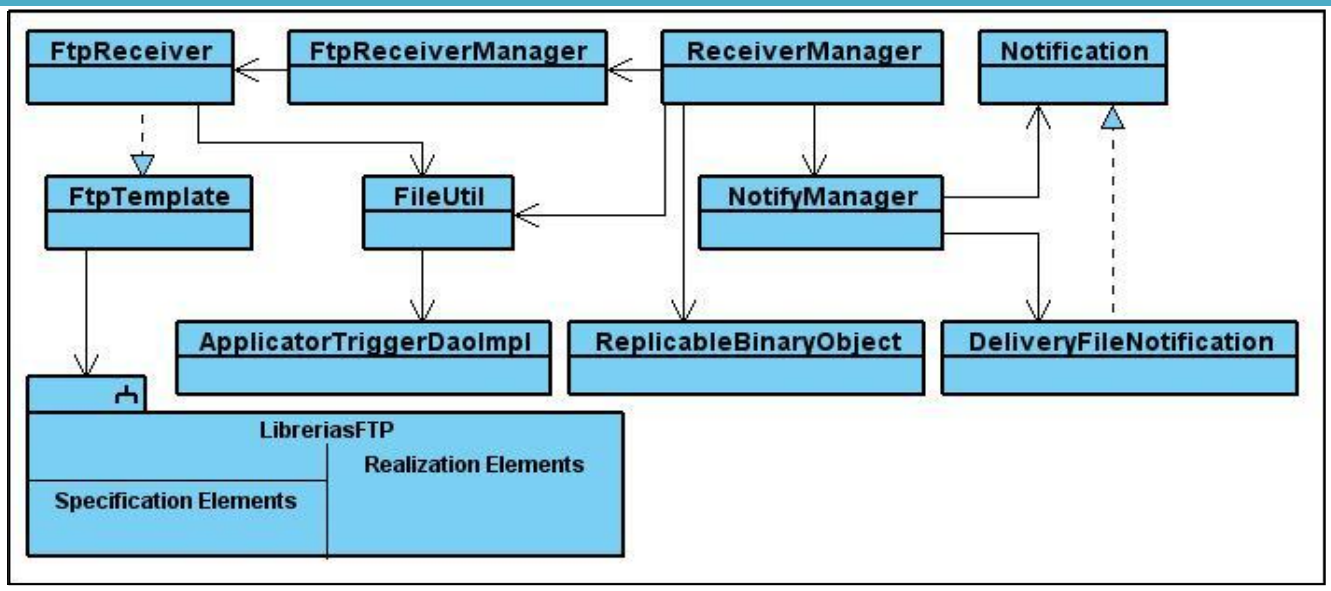


Fig. #11: Diagrama de Componentes. CU Recibir archivo binario.

4.3 Código Fuente

Clase SenderManager

```
private void convertBlobs(ReplicableGroup transaction, int numberOfTargets) {
  if (true) {
       int actionCout = 0;
       for (ReplicableAction action : transaction.getActions()) {
            int paramIndex = 0;
            for (PSParameter parameter : action.getParams()) {
                 InputStream is = null;
                 File fich = null;
                 if (parameter.getDataTypeName().equals("bytea")|| parameter.getValues()[0]
                 instanceof Blob) {
                      if (parameter.getDataTypeName().equals("bytea")) {
                           fich = (File) parameter.getValues()[0];
                           try {
                                is = new FileInputStream(fich);
                           } catch (FileNotFoundException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                           }
                      } else if (parameter.getValues()[0] instanceof Blob) {
                           try {
                                is = ((Blob) parameter.getValues()[0]).getBinaryStream(0,0);
                           } catch (SQLException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                           }
                      }
                       try {
                           GroupAction groupAction = generateFileName(actionCout,
                           transaction.getId(),paramIndex);
                           fileUtil.saveFile("part" + GeneralConfig.getNodeId() + "-" +
                           transaction.getId() + "-"
                           + paramIndex, is);
                           fich.delete();
                           fileUtil.renameFile("part" + GeneralConfig.getNodeId() + "-" +
                           transaction.getId() +                          "-" +
                           paramIndex,GeneralConfig.getNodeId() + "-" + transaction.getId() + "-"
                           + paramIndex);
                           hasTargets.put(GeneralConfig.getNodeId() + "-" + transaction.getId() +
                           "-"
                           + paramIndex, new
                           TargetsConfirmation(numberOfTargets,GeneralConfig.getNodeId() + "-"
                           + transaction.getId() + "-" + paramIndex ));
                           parameter.getValues()[0] = new
                           ReplicableBinaryObject(ReplicableBinaryObject.BLOB,false,groupAction);
                       } catch (IOException e) {
                           // TODO Auto-generated catch block
                           e.printStackTrace();
                       }
                  }
                  paramIndex++;
             }
             actionCout++;
        }
    }
  }
```

Fig. #12: Implementaciones relevantes. Clase SenderManager. Método convertBlobs.

Clase ReceiverManager

```
protected void downloadBinaryDataIfAplicable(ReplicableGroup group) {
      for (ReplicableAction action : group.getActions()) {
            for (PSParameter param : action.getParams()) {
                  System.out.println(param.toString());
                  if (param.getValues()[0] instanceof ReplicableBinaryObject) {
                        if (true) {
                              String filename = ((ReplicableBinaryObject) param.getValues()[0])
                              .getGroupAction().getSenderId()+ "-" + ((ReplicableBinaryObject) param.getValues()[0])
                              .getGroupAction().getGroupId() + "-" + ((ReplicableBinaryObject) param.getValues()[0])
                              .getGroupAction().getParamIndex();
                              ftpReceiverManager.addToDownloadList(((ReplicableBinaryObject) param.getValues()[0])
                              .getGroupAction());
                              ftpReceiverManager.beginDownload();
                              Notification not = new DeliveryFileNotification(Notification.JUST_RECEIVED,
                              transactionProcessor.getIdNode(), group.getSenderId(), group.getId(), filename);
                              not.setFilename(filename);
                              deliveryManager.sendNotification(not);
                              logger.info("Enviada confirmacion de fichero recibido");
                              InputStream is =  null;
                              try {
                                    is = fileut.readFile(filename);
                              } catch (FileNotFoundException e1) {
                                    // TODO Auto-generated catch block
                                    e1.printStackTrace();
                              }
                              if (param.getDataTypeName().equals("bytea") || (param.getValues()[0] instanceof Blob)) {
                                    param.getValues()[0] = filename;
                              }
                        }
                  }
            }
      }
}
```

Fig. #13: Implementaciones relevantes. Clase ReceiverManager. Método downloadBinaryDataIfAplicable.

4.4 Modelo de Pruebas

Las Pruebas son una actividad, en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos específicos, los resultados son observados y registrados para una posterior evaluación. Son un elemento de suma importancia para la Calidad del Software, por lo que se llevan a cabo durante todo el ciclo de vida del mismo. Constituyen una revisión final de las especificaciones del Diseño y de la Codificación.

Son realizadas con diferentes objetivos y en diferentes niveles de trabajo. De los seis niveles de trabajo existentes, se realizarán pruebas al módulo en cinco de ellos:

Prueba de desarrollador: Es la prueba diseñada e implementada por el equipo de desarrollo.

Prueba de unidad: Es la prueba enfocada a los elementos testeables más pequeño del software. Es 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.

Prueba de integración: Es ejecutada para asegurar que los componentes en el modelo de implementación operen correctamente cuando son combinados para ejecutar un Caso de Uso. El objetivo es tomar los componentes probados en unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.

Prueba de regresión: Siempre que se añade un nuevo módulo como parte de una prueba de integración, el software cambia. La prueba de regresión es la actividad que ayuda a asegurar que los cambios no introducen un comportamiento no deseado o errores adicionales. Dentro del conjunto de pruebas de regresión, se aplicarán Casos de Prueba que se centran en los componentes del software que ha cambiado.

Prueba de sistema: Son las pruebas que se hacen cuando el software está funcionando como un todo. Es la actividad de prueba dirigida a verificar el programa final, después que todos los componentes de software y hardware han sido integrados. Específicamente se realizarán pruebas de rendimiento, con el objetivo de probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado.

Se realizarán también pruebas de integridad (enfocada a la resistencia a fallos), stress (enfocada a observar cómo el sistema responde ante condiciones anormales) y configuración (enfocada a asegurar que funciona en diferentes configuraciones de hardware).

Existen dos Métodos de Prueba: Caja Blanca y Caja Negra. El primero de los métodos se basa en comprobar los caminos lógicos del software proponiendo Casos de Prueba donde se ejercitan conjuntos específicos de condiciones y/o bucles; con este se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado. El segundo método, se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. Los Casos de Prueba diseñados, pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene.

Al módulo para la trasmisión de datos de gran tamaño en el software Reko, se le harán pruebas siguiendo el método de **Caja Blanca**, pues no posee interfaces. La técnica a emplear será **Camino Básico**. Esta técnica permite obtener una medida de la complejidad lógica de un diseño procedimental y utiliza esa medida como guía para la definición de un conjunto básico de caminos de ejecución, de los cuales se obtienen los Casos de Prueba, que garantizan la ejecución de cada sentencia del programa al menos una vez, durante las pruebas.

4.4.1 Caja Blanca. Camino Básico

Para la ejecución de esta técnica existen cuatro pasos básicos:

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

Camino Básico. Clase SenderManager. Método convertBolbs

Enumeración del Código Fuente

```
private void convertBlobs(ReplicableGroup transaction, int numberOfTargets) {
1    if (true) {
2        int actionCount = 0;
3        for (ReplicableAction action : transaction.getActions()) {
4            int paramIndex = 0;
5            for (PSParameter parameter : action.getParams()) {
6                InputStream is = null;
6                File fich = null;
7 y 8            if (parameter.getDataTypeName().equals("bytea")|| parameter.getValues()[0] instanceof Blob) {
9                    BinaryData blob = null;
10                   if (parameter.getDataTypeName().equals("bytea")) {
11                       fich = (File) parameter.getValues()[0];
12                       try {
12                           is = new FileInputStream(fich);
13                       } catch (FileNotFoundException e) {
                             // TODO Auto-generated catch block
13                           e.printStackTrace();
                         }
14                   } else if (parameter.getValues()[0] instanceof Blob) {
14                       try {
14                           is = ((Blob) parameter.getValues()[0]).getBinaryStream(0,0);
15                       } catch (SQLException e) {
                             // TODO Auto-generated catch block
15                           e.printStackTrace();
                         }
                     }
16                   try {
16                       GroupAction groupAction = generateFileName(actionCount, transaction.getId(),paramIndex);
16                       fileUtil.saveFile("part" + GeneralConfig.getNodeId() + "-" + transaction.getId() + "-"
                         + paramIndex, is);
16                       fich.delete();
16                       fileUtil.renameFile("part" + GeneralConfig.getNodeId() + "-" + transaction.getId() +
                         "-" + paramIndex,GeneralConfig.getNodeId() + "-" + transaction.getId() + "-"
                         + paramIndex);
16                       hasTargets.put(GeneralConfig.getNodeId() + "-" + transaction.getId() + "-"
                         + paramIndex, new TargetsConfirmation(numberOfTargets,GeneralConfig.getNodeId() + "-"
                         + transaction.getId() + "-" + paramIndex ));
                         parameter.getValues()[0] = new ReplicableBinaryObject(ReplicableBinaryObject.BLOB,
16                       false,groupAction);
17                   } catch (IOException e) {
                         // TODO Auto-generated catch block
17                       e.printStackTrace();
                     }
                 }
18               paramIndex++;
             }
19           actionCount++;
         }
     }
20 }
```

Fig. #14: Técnica Camino Básico. Paso #1.

Construcción del grafo correspondiente al Código Fuente

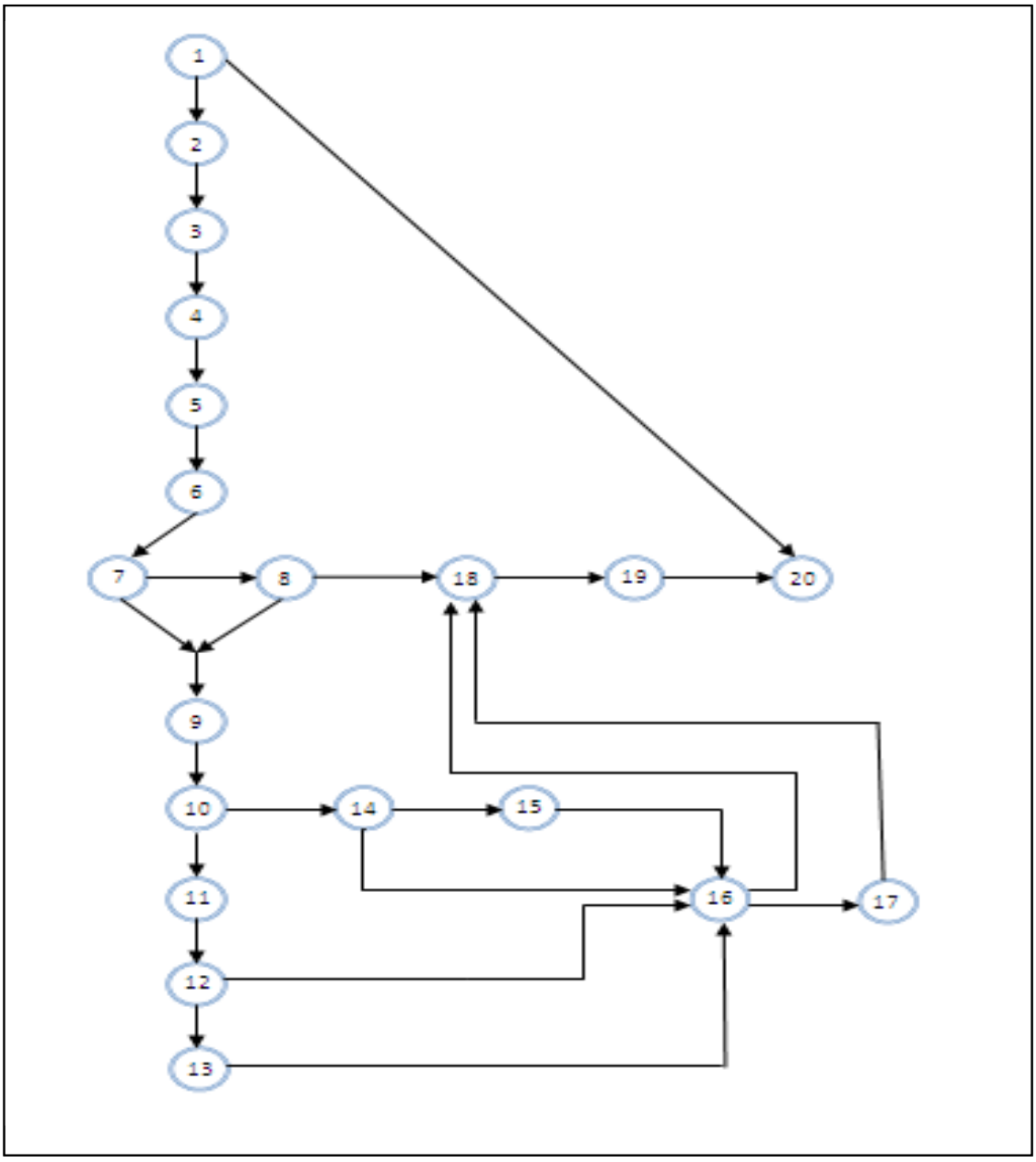


Fig. #15: Técnica Camino Básico. Paso #1.

Paso #2: Complejidad Ciclomática

Proporciona una medición cuantitativa de la complejidad lógica de un programa, define el número de caminos independientes del conjunto básico de este y brinda el límite superior para el número de pruebas a realizar que aseguren la ejecución de todas las sentencias al menos una vez. La Complejidad Ciclomática se denota por *V* y una de las vías utilizadas para calcularla es $V(G) = P + 1$. Donde, *P* es el número de nodos predicado (nodo de donde salen más de una arista) del grafo G.

$V(G) = P + 1$

$V(G) = 7 + 1$

$V(G) = 8$

Paso #3: Determinar un conjunto de Caminos Básicos Independientes

Camino #1: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 18 – 19 – 20.

Camino #2: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 14 – 16 – 18 – 19 – 20.

Camino #3: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 14 – 15 – 16 – 18 – 19 – 20.

Camino #4: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 11 -12 – 16 – 18 – 19 – 20.

Camino #5: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 11 – 12 – 13 – 16 – 18 – 19 – 20.

Camino #6: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 11 – 12 – 16 – 17 – 18 – 19 – 20.

Camino #7: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 14 – 16 – 17 – 18 – 19 – 20.

Camino #8: 1 – 20.

Paso #4: Casos de Prueba

Caja Blanca. Método convertBlobs. Caso de Prueba #1	
Camino:	1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 18 – 19 – 20.
Caso de Prueba:	Comprobar que el parámetro sea de tipo Bytea o que sea una instancia de Blob.
Entrada:	Transaction, numberOfTargets.
Resultado:	Incrementa el paramIndex.
Condiciones:	No existe ningún dato de tipo Bytea o Blob.

Caja Blanca. Método convertBlobs. Caso de Prueba #2	
Camino:	1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 14 – 16 – 18 – 19 – 20.
Caso de Prueba:	Tipo de dato es instancia de Blob.
Entrada:	Transaction, numberOfTargets.
Resultado:	Procede a guardar el fichero.
Condiciones:	El tipo de dato del parámetro es instancia de Blob.

Caja Blanca. Método convertBlobs. Caso de Prueba #3	
Camino:	1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 14 – 15 – 16 – 18 – 19 – 20.
Caso de Prueba:	Error al instanciar el objeto con Blob.
Entrada:	Transaction, numberOfTargets.
Resultado:	Tratamiento de excepciones.
Condiciones:	Ocurre una excepción de entrada/salida.

Caja Blanca. Método convertBlobs. Caso de Prueba #4	
Camino:	1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 11 -12 – 16 – 18 – 19 – 20.
Caso de Prueba:	Tipo de dato es Bytea.
Entrada:	Transaction, numberOfTargets.
Resultado:	Procede a guardar el fichero.
Condiciones:	El tipo de dato del parámetro es Bytea.

Caja Blanca. Método convertBlobs. Caso de Prueba #5	
Camino:	1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 11 – 12 – 13 – 16 – 18 – 19 – 20.
Caso de Prueba:	Error al comprobar si el nombre del tipo de dato es igual a Bytea.
Entrada:	Transaction, numberOfTargets.
Resultado:	Tratamiento de excepciones.
Condiciones:	Ocurre una excepción de entrada/salida.

Caja Blanca. Método convertBlobs. Caso de Prueba #6	
Camino:	1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 11 – 12 – 16 – 17 – 18 – 19 – 20.
Caso de Prueba:	Error al guardar un archivo correspondiente a un tipo de dato Bytea.
Entrada:	Transaction, numberOfTargets.
Resultado:	Tratamiento de excepciones.
Condiciones:	Ocurre una excepción de entrada/salida.

Caja Blanca. Método convertBlobs. Caso de Prueba #7	
Camino:	1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 14 – 16 – 17 – 18 – 19 – 20.
Caso de Prueba:	Error al guardar un archivo correspondiente a un tipo de dato Blob.
Entrada:	Transaction, numberOfTargets.
Resultado:	Tratamiento de excepciones.
Condiciones:	Ocurre una excepción de entrada/salida.

Caja Blanca. Método convertBlobs. Caso de Prueba #8	
Camino:	1 – 20.
Caso de Prueba:	Ejecución de la responsabilidad convertBlobs.
Entrada:	Transaction, numberOfTargets.
Resultado:	No se ejecuta la responsabilidad convertBlobs.
Condiciones:	La variable booleana igual a ¨False¨.

Luego de haber sido ejecutados cada uno de los Casos de Pruebas, proporcionados por los caminos independientes, se pudo constatar cómo responderá la responsabilidad convertBlobs ante diferentes condiciones.

Los resultados obtenidos coinciden con los resultados esperados, por lo que se puede concluir que la responsabilidad responde a los requerimientos planteados.

4.5 Conclusiones

En este capítulo se presentó el Modelo de Implementación, conformado por los Diagramas de Componentes correspondientes a los Casos de Uso Enviar archivo binario y Recibir archivo binario. Se mostraron algunas implementaciones de relevancia para el desarrollo de la investigación. Para validar la completitud de respuesta a los Requerimientos, antes identificados, se realizaron pruebas al código, quedando elaborado de esta forma el Modelo de Pruebas, conformado por Casos de Prueba que se corresponden con los Casos de Uso Enviar archivo binario y Recibir archivo binario.

CONCLUSIONES

Al utilizar Reko para enviar datos de gran tamaño, se hizo evidente su incapacidad para ejecutar esta tarea de forma eficiente, por lo que se planteó la necesidad de dar solución a esta problemática.

✓ El empleo de patrones de diseño, facilitó la elaboración y construcción del Modelo de Diseño, donde se tuvo en cuenta el lenguaje de programación a emplear, antes definido. Posteriormente se procedió a la Realización de los Casos de Uso del Diseño, donde se especificó la relación y la interacción existente entre las clases; y se realizó, además, una breve descripción de las clases implicadas. Posteriormente se brindo la Vista de Despliegue, donde se pudo apreciar la distribución física del sistema en términos de nodos y sus relaciones.

✓ Partiendo de los Diagramas de Clases del Diseño se elaboraron los Diagramas de Componentes que conforman el Modelo de Implementación. Luego, estos componentes fueron implementados e integrados al software Reko, para dar solución a la problemática planteada.

✓ Una vez terminada la implementación se procedió a realizar pruebas exploratorias, con el objetivo de verificar si el módulo daba respuesta a los requerimientos especificados. Los resultados obtenidos, probaron que Reko había sido dotado de mecanismos eficientes para la réplica de datos de gran tamaño.

RECOMENDACIONES

- ✓ Implementar protocolos de seguridad como SSL, para lograr la seguridad en el envío de datos mediante servidores FTP.
- ✓ Establecer dominios de réplica con el objetivo de agilizar el proceso de transferencia de datos.
- ✓ Implementar mecanismos que permitan el proceso de réplica para sistemas de archivos.

REFERENCIAS BIBLIOGRÁFICAS

[1]. *Oracle® Database Advanced Replication 11g Release 2 (11.2) E10706-02.* [En línea] Diciembre de 2009. [Citado el: 08 de Febrero de 2010.] [Disponible en: http://download.oracle.com/docs/cd/E11882_01/server.112/e10706.pdf.].

[2]. *Daffodil Software.* [En línea] © 2009 Daffodil Software. [Citado el: 08 de Febrero de 2010.] [Disponible en: http://enterprise.replicator.daffodilsw.com/.].

[3]. *Continuent.* [En línea] [Citado el: 09 de Febrero de 2010.] [Disponible en: http://www.continuent.com/downloads/documentation.].

[4]. **Tilma, B. J.** *BD Replicador.* 2008e.

[5]. **SYMMETRICDS.** *SymmetricDS User Guide.* 2007d.

[6]. *UCIForge | Entorno Virtual de Desarrollo Colaborativo.* [En línea] UCI. [Citado el: 07 de Febrero de 2010.] [Disponible en: https://forge.uci.cu/gf/project/reko/.].

[7]. **Jordana, Garcilaso.** *Introducción Open UP.* [En línea] Octubre de 2008. [Citado el: 10 de Febrero de 2010.] [Disponible en: http://www.mug.org.ar/Descargas/Jornadas/Downloads_GetFile.aspx?id=3136.].

[8]. **Balduino, Ricardo.** *Introduction to OpenUP (Open Unified Process).* [En línea] Agosto de 2007. [Citado el: 09 de Febrero de 2010.] [Disponible en: http://www.eclipse.org/epf/general/OpenUP.pdf.].

[9]. *Virtual Formac.* [En línea] 2010. [Citado el: 11 de Febrero de 2010.] [Disponible en: http://www.virtual-formac.com/curso-lenguaje-de-modelado-uml-c5024.html.].

[10] Osmasis Latina. [En línea] 31 de Diciembre de 2007. [Citado el: 11 de Febrero de 2010.][Disponible en: http://www.osmosislatina.com/lenguajes/uml/basico.htm].

[11] Perissé, Marcelo Claudio. Proyecto Informático / Una Metodología Simplificada. Ciencia y Técnica Administrativa. [En línea] [Citado el: 12 de Febrero de 2010.] [Disponible en: http://www.cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro/c5/c5.htm.].

[12] Giraldo, Luis y Zapata, Yuliana. Herramientas de Desarrollo de Ingeniería de SW para LINUX. [En línea] 24 de Septiembre de 2005.[Citado el: 12 de Febrero de 2010.] [Disponible en: http://hugolopez.phi.com.co/docs/download/file=Giraldo-Zapata-Herramientas%20de%20ISW.pdf,_id=17.].

[13] Historia del lenguaje Java. [En línea] Computación Aplicada al Desarrollo SA de CV.[Disponible en: http://www.cad.com.mx/historia_del_lenguaje_java.htm.] .

[14] Sanz, Laura Bermejo y Monreal, Enrique Gómez.Eclipse como IDE.[En línea][Citado el: 10 de Febrero de 2010.] [Disponible en: http://kybele.escet.urjc.es/documentos/HC/Exposiciones/EclipseIDE.pdf.].

[15] JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. "El Proceso Unificado de Desarrollo de Software".2000. Addison Wesley. Capítulos 7, 8 páginas 125-163, 187-202. [Citado el: 20 de Febrero de 2010].

[16] **Szyperski, Clemens.** *Component software: Beyond object-oriented programming.* s.l. : Addison-Wesley Pub Co, 2da edición, Noviembre 2002.

[17] **Gamma, Erich, y otros.** *Design Patterns. Elements of Reusable Object-Oriented Software.* s.l. : Addison Wesley , 1995. ISBN 0-201-63361-2.

[18] **Saavedra Gutierrez, Jorge A.** El Mundo Informatico. [En línea] 17 de Agosto de 2006. [Citado el: 25 de Abril de 2010.] [Disponible en: http://jorgesaavedra.wordpress.com/2006/08/17/patrones-grasp-craig-larman/].

[19] **Visconti, Marcello y Astudillo, Hernán.** Fundamentos de Ingeniería de Software. *Departamento de Informática. Univerdad Técinca Federico Santa María.* [En línea] [Citado el: 25 de Abril de 2010.] [Disponible en: http://www.inf.utfsm.cl/~visconti/ili236/Documentos/15-Implementacion.pdf].

[20] **JACOBSON, I.; G. BOOCH**, *et al. El proceso Unificado de Desarrollo de Software*. p.xviii *Addison Wesley Object Technology.* 84-7829-036-2

[21] **Visconti, Marcello y Hernán, Astudillo.** Departamento de Informática. [En línea] [Citado el: 29 de Abril de 2010.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/15-Implementacion.pdf.

BIBLIOGRAFÍA

1. Asociación Antioqueña de Ingeniería Informática. [En línea] [Citado el: 10 de Febrero de 2010.] [Disponible en: http://www.willydev.net/descargas/prev/Prologo.pdf].

2. **Balduino, Ricardo.** *Introduction to OpenUP (Open Unified Process).* [En línea] Agosto de 2007. [Citado el: 09 de Febrero de 2010.] [Disponible en: http://www.eclipse.org/epf/general/OpenUP.pdf].

3. *Cientec.* [En línea] [Citado el: 11 de Febrero de 2010.] [Disponible en: http://www.cientec.com/analisis/ana-uml.html].

4. *Continuent.* [En línea] [Citado el: 09 de Febrero de 2010.] [Disponible en: http://www.continuent.com/downloads/documentation].

5. *Daffodil Software.* [En línea] © 2009 Daffodil Software. [Citado el: 08 de Febrero de 2010.] [Disponible en: http://enterprise.replicator.daffodilsw.com/].

6. **Fuentes, Lidia; Troya, José M. y Vallecillo, Antonio.** *Desarrollo del Software Basado en Componentes.* [En línea] [Citado el: 10 de Marzo de 2010.] [Disponible en: http://www.lcc.uma.es/~av/Docencia/Doctorado/tema1.pdf].

7. **Gamma, Erich, y otros.** *Design Patterns. Elements of Reusable Object-Oriented Software.* s.l. : Addison Wesley , 1995. ISBN 0-201-63361-2.

8. **Giraldo, Luis y Zapata, Yuliana.** *Herramientas de Desarrollo de Ingeniería de SW para LINUX.* [En línea] 24 de Septiembre de 2005. [Citado el: 12 de Febrero de 2010.] [Disponible en: http://hugolopez.phi.com.co/docs/download/file=Giraldo-Zapata-Herramientas%20de%20ISW.pdf,_id=17].

9. **Gracia Murugarren, Joaquín.** Ingeniero Software. Análisis y Diseño. [En línea] 2003. [Citado el: 26 de Abril de 2010.] [Disponible en: http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php].

10. *Historia del lenguaje Java.* [En línea] Computación Aplicada al Desarrollo SA de CV. [Citado el: 12 de Febrero de 2010.] [Disponible en: http://www.cad.com.mx/historia_del_lenguaje_java.htm.]

11. **JACOBSON, I.; G. BOOCH**, *et al. El proceso Unificado de Desarrollo de Software*. p. *Addison Wesley Object Technology.* 84-7829-036-2

12. **Jordana, Garcilaso.** *Introducción Open UP.* [En línea] Octubre de 2008. [Citado el: 10 de Febrero de 2010.] [Disponible en: http://www.mug.org.ar/Descargas/Jornadas/Downloads_GetFile.aspx?id=3136].

13. **Kroll, Per y MacIsaac, Bruce.** *Agility and Discipline Made Easy. Practices from PenUp and Rup.* s.l. : Addison- Wesley, 2006.

14. *Open Up.* [En línea] última actualización: 17 de Agosto de 2009. [Citado el: 10 de Febrero de 2010.] [Disponible en: http://epf.eclipse.org/wikis/openup/].

15. *Oracle® Database Advanced Replication 11g Release 2 (11.2) E10706-02.* [En línea] Diciembre de 2009. [Citado el: 08 de Febrero de 2010.] [Disponible en http://download.oracle.com/docs/cd/E11882_01/server.112/e10706.pdf.].

16. *Osmasis Latina.* [En línea] 31 de Diciembre de 2007. [Citado el: 11 de Febrero de 2010.] [Disponible en: http://www.osmosislatina.com/lenguajes/uml/basico.htm].

17. **Palliotto, Diana y Romano, Gabriel.** *¿Cuáles son las características que debe tener una herramienta UML?* [En línea] [Citado el: 09 de Febrero de 2010.] [Disponible en: http://www.docirs.cl/caracteristica_herramienta_uml.htm].

18. **Perissé, Marcelo Claudio.** Proyecto Informático / Una Metodología Simplificada. *Ciencia y Técnica Administrativa.* [En línea] [Citado el: 12 de Febrero de 2010.] [Disponible en : http://www.cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro/c5/c5.htm].

19. **Saavedra Gutierrez, Jorge A.** El Mundo Informatico. [En línea] 17 de Agosto de 2006. [Citado el: 25 de Abril de 2010.] [Disponible en: http://jorgesaavedra.wordpress.com/2006/08/17/patrones-grasp-craig-larman/].

20. **Sanz, Laura Bermejo y Monreal, Enrique Gómez.** *Eclipse como IDE.* [En línea] [Citado el: 10 de Febrero de 2010.] [Disponible en: http://kybele.escet.urjc.es/documentos/HC/Exposiciones/EclipseIDE.pdf]

21. *Sitio Oficial Eclipse.* [En línea] The Eclipse Foundation. [Citado el: 12 de Febrero de 2010.] [Disponible en: http://www.eclipse.org/].

22. **Szyperski, Clemens.** *Component software: Beyond object-oriented programming.* s.l. : Addison-Wesley Pub Co, 2da edición, Noviembre 2002.

23. **The ProFTPD Project. Pr**oFTPD. [En línea] 5 de Febrero de 2009. [Citado el: 15 de Febrero de 2010.] [Disponible en: http://www.proftpd.org/].

24. *UCIForge | Entorno Virtual de Desarrollo Colaborativo.* [En línea] UCI. [Citado el: 07 de Febrero de 2010.] [Disponible en: https://forge.uci.cu/gf/project/reko/].

25. *Virtual Formac.* [En línea] 2010. [Citado el: 11 de Febrero de 2010.] [Disponible en: http://www.virtual-formac.com/curso-lenguaje-de-modelado-uml-c5024.html].

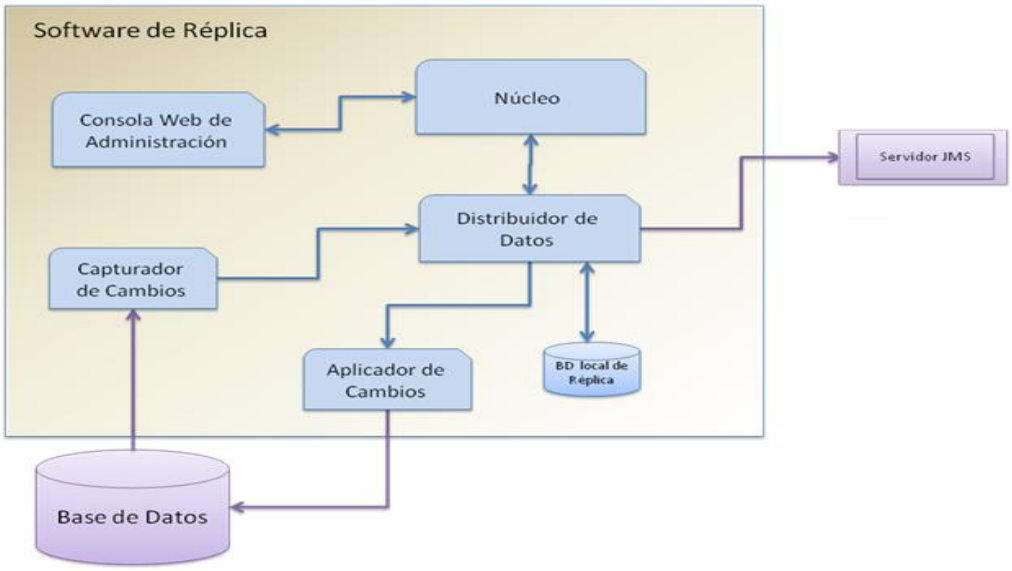
26. **Visconti, Marcello y Astudillo, Hernán.** *Fundamentos de Ingeniería de Software*. Departamento de Informática. Univerdad Técinca Federico Santa María. [En línea] [Citado el: 25 de Abril de 2010.] [Disponible en: http://www.inf.utfsm.cl/~visconti/ili236/Documentos/15-Implementacion.pdf].

27. **Visco*nti, Marcello y Hernán, Astudillo.*** *Departamento de Inf*ormática*. [En línea] [Citado el: 29 de A*bril de 2010.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/15-Implementacion.pdf.

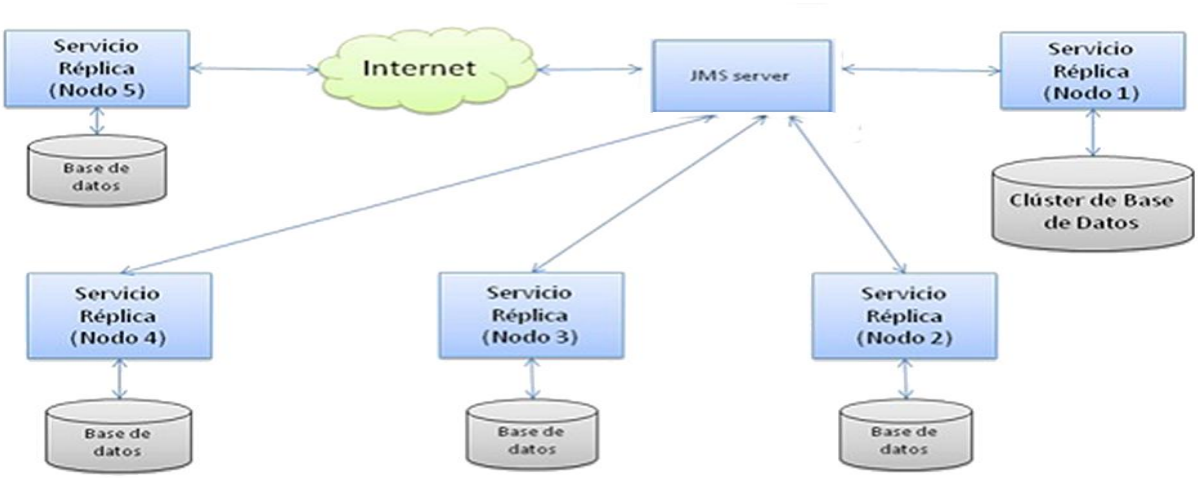
28. **Zamitiz, Ing. Carlos Alberto Román.** *Temas Especiales de Computación.* [En línea] [Citado el: 10 de Febrero de 2010.] [Disponible en: http://profesores.fi-b.unam.mx/carlos/aydoo/toc.html].

ANEXOS

Anexo #1: Principales componentes del software de Réplica Reko.



Anexo #2: Ejemplo de Escenario de Replicación.



GLOSARIO

Publicador: es una instancia de base de datos que permite que los datos estén disponibles para otras ubicaciones a través de la réplica. El publicador puede tener una o más publicaciones, cada una de las cuales representa un conjunto de datos de objetos relacionados lógicamente para su réplica.

Subscriptor: es una instancia de base de datos que recibe datos replicados. Un subscriptor puede recibir datos de varios publicadores y publicaciones. En función del tipo de réplica elegida, el subscriptor también puede devolver los datos modificados al publicador o volver a publicar los datos en otros subscriptores.

Nodo: unidad de replicación que incluye una base de datos y una instancia del software Reko.

Etiqueta: conjunto de nodos agrupados, con el objetivo de aplicar una misma configuración sobre estos.

Triggers: también conocido como desencadenador, éste se ejecuta des atendidamente y automáticamente cuando un usuario realiza una acción con la tabla de una base de datos que lleve asociado esta operación (Operaciones DML).

Log: también conocido como bitácora, es un archivo que registra movimientos y actividades de un programa determinado (log file).

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