UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 4

Centro de Consultoría y Desarrollo de Arquitecturas Empresariales

Mecanismo de réplica de ficheros para el Replicador de datos REKO v5.0

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

Autor: Rogelio Daniel González Martínez

Tutor: Ing. Enrique Basto Muguercia

La Habana, Julio de 2017

"Año 59 de la Revolución"

“Las inteligencias poco capaces se interesan en lo extraordinario; las inteligencias poderosas, en las cosas ordinarias."

Víctor Hugo

AGRADECIMIENTOS

A mis padres por dar a luz a una persona tan buena como yo, por pasar malos y buenos momentos junto a mí, gracias por aconsejarme y guiarme en el camino de la vida.

A mi familia por su apoyo incondicional y confianza en todo momento de la carrera.

A mi tía, que es como mi segunda madre que siempre ha estado pendiente de mí en todos estos años de estudios y preocupada que nunca me haya faltado nada.

Mi hermanita que es la luz de mis ojos que la quiero un montón.

Mi tío, mis primos siempre pendientes de cuando voy a la casa o no y preocupados por como estoy y que necesito para darme todo el apoyo y ayuda posible.

A mi tutor por la gran ayuda que me brindó durante el desarrollo de este trabajo, sin él no hubiera conseguido este sueño que hoy en día vivo de ser Ingeniero.

A todos mi amigos y compañeros del aula, a todos mis compañeros del dota, del futbol y del edificio, la pandilla del 91102 que compartimos este año grandes historias.

A todo aquel que de una forma u otra ha influido en mi vida a pasar buenos o malos momentos en estos años aquí en la UCI, los que me hayan deseado mal también me ayudaron a superarme y hacerlos quedar en ridículo.

DEDICATORIA

A mi familia por siempre estar a mi lado y apoyarme en toda decisión que tomaba y en los peores momentos decirme, no pasa nada, a seguir para delante.

Muy especial a dos personas que ya no tengo conmigo hace un tiempo , mi abuela María, mi primera madre, encargada de criarme y consentirme en mis primeros caprichos y malcriadeces, hasta en tus últimos momentos cuidándome y diciendo dale cositas al niño, nunca olvidaré esos momentos, siempre estarás conmigo y si hoy soy ingeniero, mucho te lo debo a ti, gracias. Mi abuelo Pancho que la última alegría que le pude dar de mi parte es que había ingresado a la Universidad de las Ciencias Informáticas, orgulloso de mi por haber cogido una carrera universitaria, si hoy estuvieras aquí estarías más orgulloso de que me hice ingeniero, de que conseguí realizar el ultimo sueño que supiste mío, mucho luché, muchos trabajos y contratiempos, pero logré el objetivo.

DECLARACIÓN JURADA DE AUTORÍA

Declaro ser el único autor de la presente tesis y reconozco 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 _____.

_____ _____

Rogelio D. González Martínez
Autor

Ing.Enrique Basto Muguercia
Tutor

RESUMEN

En la presente investigación se pretende mejorar la versión del Replicador Reko desarrollado en el Centro de Consultoría y Desarrollo de Arquitecturas Empresariales ubicado en la Universidad de las Ciencias Informáticas.

El replicador de datos Reko es un software multiplataforma que permite realizar la réplica de datos entre distintos gestores de bases de datos, cuenta dentro de sus funcionalidades con un mecanismo que se encarga de realizar la réplica de fichero. Actualmente este mecanismo presenta limitantes durante el envío de ficheros, debido a que utiliza como medio de comunicación el servidor de mensajería ActiveMQ, servidor que es una implementación del estándar JMS, el cual no soporta nativamente el envío de ficheros. El presente trabajo de diploma propone un mecanismo para el proceso de réplica de ficheros, que permita disminuir las inconsistencias entre las bases de datos durante el proceso de réplica. Para el desarrollo de la solución se describe el problema inicial del cual surgió el mecanismo propuesto, el uso de los métodos teóricos y empíricos, así como un análisis de los mecanismos de comunicación entre ordenadores y framework que basan su funcionamiento en la comunicación mediante sockets. Se hizo uso de la metodología AUP para guiar el diseño e implementación del mecanismo.

PALABRAS CLAVE:

base de datos,distribuidor,replicador,comunicación socket,netty

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Actualmente, con el auge de las tecnologías, se ha propiciado la necesidad de contar con aplicaciones que sean capaces de manejar grandes volúmenes de información, mayormente proveniente de los usuarios. Esta información necesita ser almacenada para su uso posterior e interacción con las aplicaciones que necesiten de ella. Una de las formas más utilizadas para esto es mediante los Sistemas de Bases de Datos Distribuidos (SBDD), los cuales se componen de bases de datos geográficamente dispersas, comúnmente llamadas nodos. Una de las principales características que posee los SBDD es la de mantener su información siempre actualizada, para lograr esto, se hace uso de la réplica de datos.

La réplica de datos es muy usada en aplicaciones informáticas ya que permite el acceso fácil a la información desde distintas localidades o nodos de trabajo de la red de computadoras. Esta consiste en tener la misma información en varios ordenadores y que cada una de ellas tenga total independencia de las demás ante la ocurrencia de fallos. Además, utiliza herramientas de software especializadas que detectan cambios en el sistema distribuido, uno de los más usados son los replicadores de datos.

El replicador de datos REKO, es un software de réplica de datos entre bases de datos distribuidas desarrollado en la Universidad de las Ciencias Informáticas por un grupo perteneciente al Centro de Consultoría y Desarrollo de Arquitecturas Empresariales, que brinda una herramienta multiplataforma que le permite al usuario con el menor esfuerzo cubrir las necesidades fundamentales de replicación de datos. Cuenta con un mecanismo que permite durante el proceso de réplica, el envío y recepción de los ficheros que se encuentran almacenados o referenciados en las bases de datos, garantizando la consistencia de las bases de datos que intervienen en el proceso de réplica. El mecanismo utiliza el servidor de mensajería ActiveMQ como medio de comunicación para realizar el envío de los ficheros de una instancia del replicador a otra. Este utiliza un sistema de cola para realizar el envío, el cual no envía la siguiente solicitud hasta que la anteriormente enviada no haya sido finalizada, por lo que, en caso de interrupciones, una solicitud puede estar en cola un tiempo indefinido. Además, el ActiveMQ es una implementación del protocolo Java Message Service (JMS), el cual no soporta el envío de ficheros de gran tamaño. Sin embargo, actualmente como alternativa para realizar el envío, se fraccionan los archivos en pequeños pedazos que son enviados hacia su destino donde son unidos nuevamente, acción que no es soportada por todo tipo de ficheros, lo

que provoca en algunos casos que existan ficheros corruptos en los nodos destinos y por tanto surjan inconsistencias entre las bases de datos que intervienen en el proceso de réplica.

Luego de algunas revisiones del mecanismo de réplica de ficheros por parte del equipo de proyecto del replicador de datos REKO, se identificó que algunas de las funcionalidades implementadas, en particular la réplica de ficheros no cumple las necesidades del todo al equipo de proyecto para la nueva versión del replicador.

Por lo anterior se define el siguiente **problema**: ¿Cómo garantizar la réplica de ficheros, en el replicador de datos REKO v5.0 de manera que se elimine la inconsistencia en los datos replicados?

El **objeto de estudio**, se enmarca en los mecanismos de comunicación entre ordenadores.

Se define como **campo de acción**, el proceso de réplica de ficheros utilizando mecanismos de comunicación entre ordenadores.

Para solucionar el problema planteado se define como **objetivo general** del presente trabajo de diploma: Desarrollar un mecanismo de réplica de ficheros para el replicador de datos REKO v5.0, de manera que se elimine la inconsistencia de los datos replicados.

Este objetivo general puede desglosarse en los siguientes **objetivos específicos:**

- Elaborar el marco teórico de la investigación para organizar los conocimientos científicos acumulados, los principales autores que trabajan los mecanismos de comunicación entre ordenadores y asumir posición respecto a los que usan comunicación socket.
- Desarrollar el diseño e implementación de las mejoras funcionales al mecanismo de réplica de ficheros, poniendo en práctica la metodología y herramientas seleccionadas para la obtención de un producto acorde a los requisitos funcionales propuestos.
- Validar la solución propuesta mediante la aplicación de pruebas de software para el aseguramiento de la calidad del producto obtenido.

La **idea a defender** en el presente trabajo de diploma radica en: El diseño e implementación de un mecanismo de réplica de ficheros para una disminución de los archivos corruptos en el proceso de réplica del replicador de datos REKO 5.0.

Posibles resultados:

Elaborar un informe detallado con toda la base teórica-práctica con respecto a los mecanismos de comunicación entre ordenadores sobre la cual se sustenta la solución propuesta.
Elaborar un mecanismo de comunicación para la réplica de ficheros para el replicador REKO, con el cual se logre que la réplica de ficheros se separe de la réplica de datos como tal y disminuya la inconsistencia de los datos en dicho proceso.

Tareas de investigación

- Realización del marco conceptual para precisar los principales conceptos que se emplean en la investigación.
- Elaboración del estado del arte y las herramientas que se necesiten dentro de sus procesos la obtención de metadatos, para sintetizar los resultados alcanzados en la revisión bibliográfica e indagaciones realizadas relacionadas con el tema.
- Descripción de las herramientas, tecnologías y lenguajes a utilizar para definir el ambiente de desarrollo.
- Definición de los requisitos funcionales y no funcionales para identificar las capacidades que tienen que ser alcanzadas por el sistema para cumplir los objetivos trazados.
- Descripción de la arquitectura que soporta la implementación de las funcionalidades.
- Realización del análisis y diseño del mecanismo para describir los artefactos.
- Implementación de las funcionalidades que dan cumplimiento a los requisitos identificados.
- Realización de pruebas al mecanismo en entornos de producción para valorar la calidad del producto implementado.
- Valoración de los resultados obtenidos.

Los métodos empleados en la presente investigación se describen a continuación:

Métodos teóricos

- **Analítico-Sintético:** se realizó un análisis sobre las diferentes teorías que existen en torno al concepto de mecanismos de comunicación entre ordenadores y los distintos protocolos, permitiendo sintetizar el conocimiento y dar solución al problema planteado.
- **Histórico-Lógico:** hizo posible que, mediante un estudio cronológico sobre el estado actual del desarrollo de herramientas de mecanismos de comunicación, se obtuviera la información necesaria que se aplicaría en la solución del problema.

Métodos empíricos

- **Consulta de la información:** hizo posible que, mediante una minuciosa revisión bibliográfica, se definieran los elementos esenciales que permitieran la elaboración de todos los aspectos teóricos y conceptuales de la investigación.

El presente documento consta de tres capítulos, conclusiones, recomendaciones, referencias bibliográficas y anexos.

En el **Capítulo 1. Fundamentación teórica:** se realiza un pequeño bosquejo sobre la réplica de datos, incluye el estudio de algunos mecanismos y protocolos existentes. Se selecciona la metodología de software a utilizar y las herramientas empleadas en el desarrollo de la solución.

En el **Capítulo 2. Características del sistema:** incluye la descripción, análisis y diseño de la solución que se propone para darle respuesta a la problemática planteada. Se especifican los requisitos funcionales y los no funcionales. Se realiza la descripción de los requisitos, patrones de diseño aplicados, así como los diagramas de paquetes, los diagramas de clases del diseño y el modelo de despliegue.

En el **Capítulo 3. Implementación y pruebas:** muestra la implementación de las funcionalidades. Se describen los diagramas de componentes. Se muestra el código fuente de las principales clases y se aplican pruebas al sistema para demostrar la robustez del mismo.

Capítulo 1. Fundamentación teórica

En el presente capítulo, se abordan temas referentes a la replicación de bases de datos distribuidas y conceptos fundamentales de los mecanismos de comunicación entre ordenadores y sus protocolos. Se realiza una descripción de la metodología y herramientas utilizadas, así como sus roles y artefactos.

1.1 Marco conceptual

En esta sección se ofrecen algunos de los principales conceptos que se consideran necesarios para una mejor comprensión de la investigación por parte del lector.

1.1.1 Replicación

La replicación es el proceso de copiar y administrar objetos de base de datos, tales como tablas, hacia múltiples bases de datos en localidades remotas que son parte de un Sistema de bases de datos distribuidos (SBDD). Los cambios ejecutados en una localidad son capturados y guardados localmente antes de ser aplicados a las localidades remotas. Dado que la replicación depende de una tecnología de base de datos distribuida, la replicación ofrece beneficios en las aplicaciones, que no son posibles en un ambiente puro de base de datos distribuida, tal como la disponibilidad y rendimiento [1].

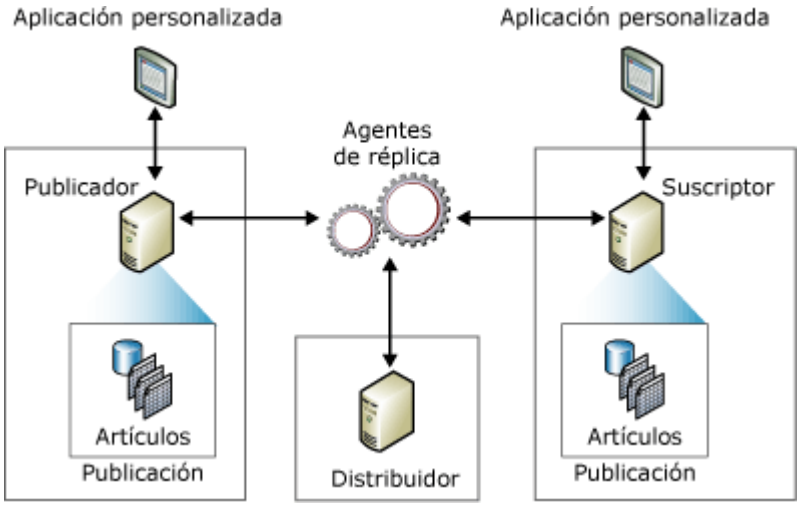


Figura 1 Ejemplo de replicación.

1.1.2 Bases de datos distribuidas

Son las que almacenan datos que pertenecen lógicamente a un solo sistema, pero se encuentra físicamente esparcido en varios “sitios” de la red. Un sistema de base de datos distribuidos se compone de un conjunto de sitios, conectados entre sí mediante algún tipo de red de comunicaciones, en el cual:

- Cada sitio es un sistema de base de datos en sí mismo.
- Los sitios trabajan en conjunto si es necesario con el fin de que un usuario de cualquier sitio pueda obtener acceso a los datos de cualquier punto de la red tal como si todos los datos estuvieran almacenados en el sitio propio del usuario [2].

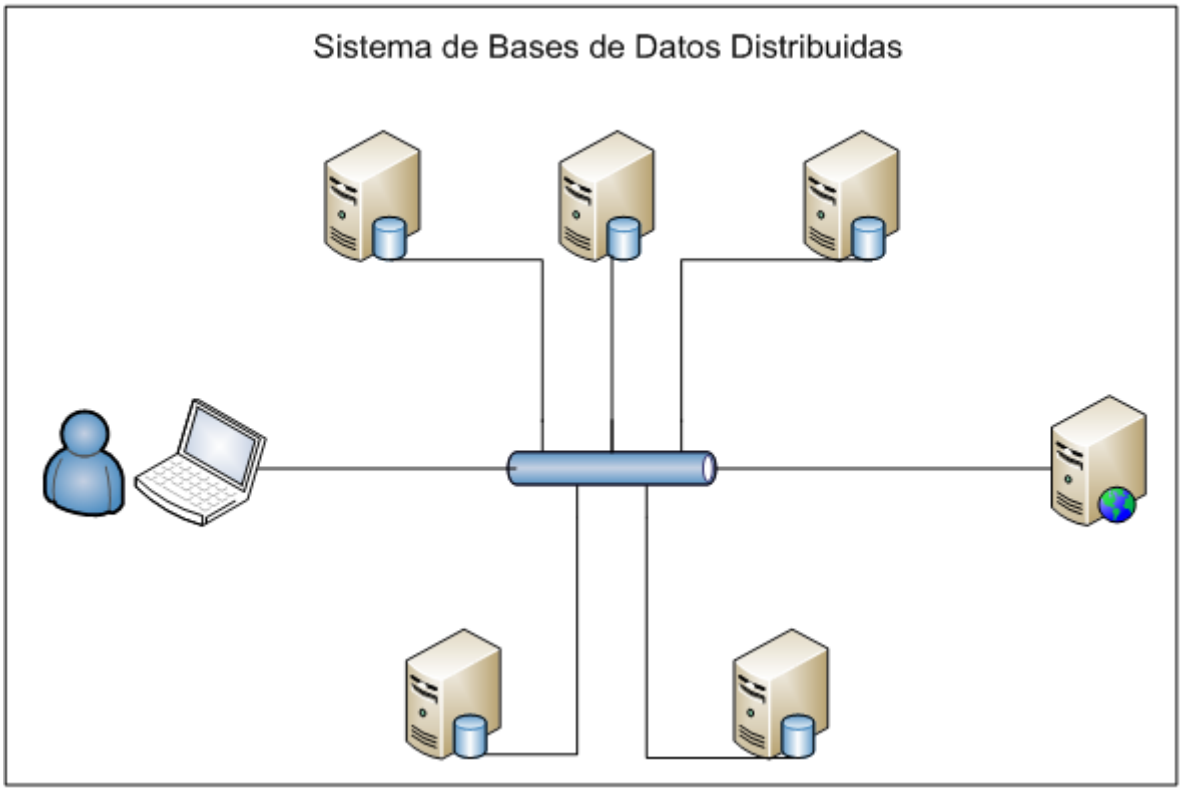


Figura 2 Ejemplo de Bases de datos distribuidas.

1.1.3 Fichero

Existen dos tipos de ficheros:

- **Ficheros ASCII o ficheros de texto:** contienen caracteres codificados según el código ASCII y se pueden leer con cualquier editor de texto como Notepad. Suelen tener extensión *.txt o *.bat, pero también otras como *.m para los programas de Matlab, *.c para los ficheros fuente de C, *.cpp para los ficheros fuente de C++, *.bas para los módulos de Visual Básic y *.java para los de Java.
- **Ficheros binarios**: son ficheros que almacenan la información de los datos o programas tal como están en la memoria del ordenador. No son legibles directamente por el usuario. Tienen la ventaja de que ocupan menos espacio en disco y que no se pierde tiempo cambiándolos a formato ASCII[1] al escribirlos y al leerlos en el disco [3].

1.1.4 Comunicación entre ordenadores

La comunicación entre computadoras es la transmisión de datos e información a través de un canal de comunicaciones entre dos computadoras, se logra mediante la utilización de redes. La red más sencilla es una conexión directa entre dos computadoras. Sin embargo, también pueden conectarse a través de grandes redes que permiten a los usuarios intercambiar datos, comunicarse mediante correo electrónico y compartir recursos [4].

[1] **ASCII** (acrónimo inglés de *American Standard Code for Information Interchange* — Código Estándar Estadounidense para el Intercambio de Información), es un código de caracteres basado en el alfabeto latino, tal como se usa en inglés moderno.

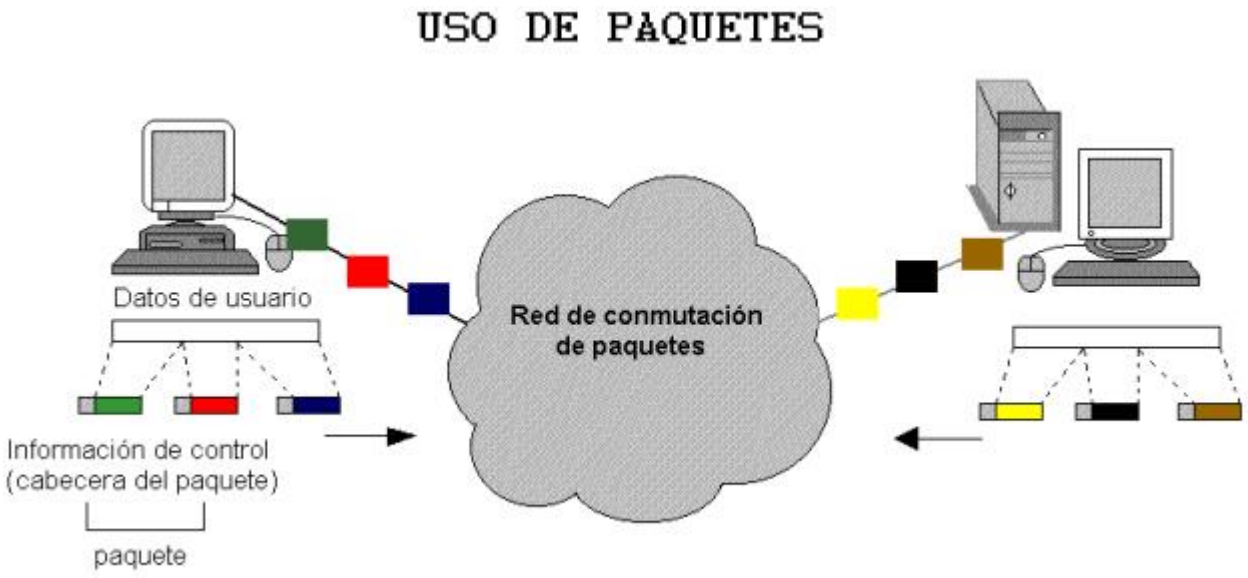


Figura 3 Comunicación entre ordenadores.

1.1.5 Mecanismos y protocolos de comunicación entre ordenadores

Los **mecanismos de comunicación entre ordenadores** permiten que se haga posible la comunicación entre procesos, contribuyen el acceso a los recursos externos de la misma forma que acceden a los recursos locales [5].

Los **protocolos de comunicación** son un conjunto de reglas que permiten la transferencia e intercambio de datos entre los distintos dispositivos que conforman una red [6].

1.2 Estudio de mecanismos

Con el objetivo de recopilar información para sintetizar los conocimientos necesarios y proponer una correcta solución a la presente investigación se realizó el análisis de algunos mecanismos de comunicación entre ordenadores y protocolos de comunicación. Existen distintos mecanismos de comunicación entre ordenadores, entre ellos:

1.2.1 Tuberías (Pipes)

Son un mecanismo para la comunicación de datos entre procesos que permiten conectar automáticamente la salida estándar de un programa con la entrada estándar en otro programa [7].

Para mejorar el rendimiento, la mayoría de los sistemas operativos implementan las tuberías utilizando espacios en memoria, lo que permite al proceso proveedor, generar más datos que los que el proceso consumidor puede atender inmediatamente. Por esta razón, son muy usadas en los sistemas operativos para garantizar la multitarea de los mismos y en lenguajes de programación como Perl, Bash, C, Python, entre otros. Las tuberías son la manera más fácil de comunicación entre procesos y son muy utilizadas mediante el símbolo "|", el cual permite comunicar dos procesos por medio de una tubería desde la terminal o línea de comando de Linux o Windows[7].

Ventaja de las tuberías:

- Además de permitir la comunicación, funcionan para transferir información entre procesos o hilos[7].

Desventajas de las tuberías:

- Son unidireccionales, es decir, el proceso que envía una salida a otro proceso, no puede a la vez, recibir una entrada de este, por lo que la comunicación que se establece es en una sola dirección.
- El envío de información que permiten es muy básico. Sólo pueden ser utilizadas en la comunicación entre procesos en un mismo ordenador[7].

1.2.2 Socket

Los sockets son un mecanismo de comunicación que constituye un componente básico de la comunicación interprocesos e intersistemas, representan un extremo de una comunicación bidireccional y tienen asociada una dirección IP (Internet Protocol o Protocolo de Internet), un protocolo y un número de puerto. Su función principal es permitir a dos programas (posiblemente situados en computadoras distintas) intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada. Su característica más importante es permitir la implementación de una arquitectura cliente-servidor, pues la comunicación es iniciada por uno de los programas que se denomina programa cliente y el segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa servidor [8].

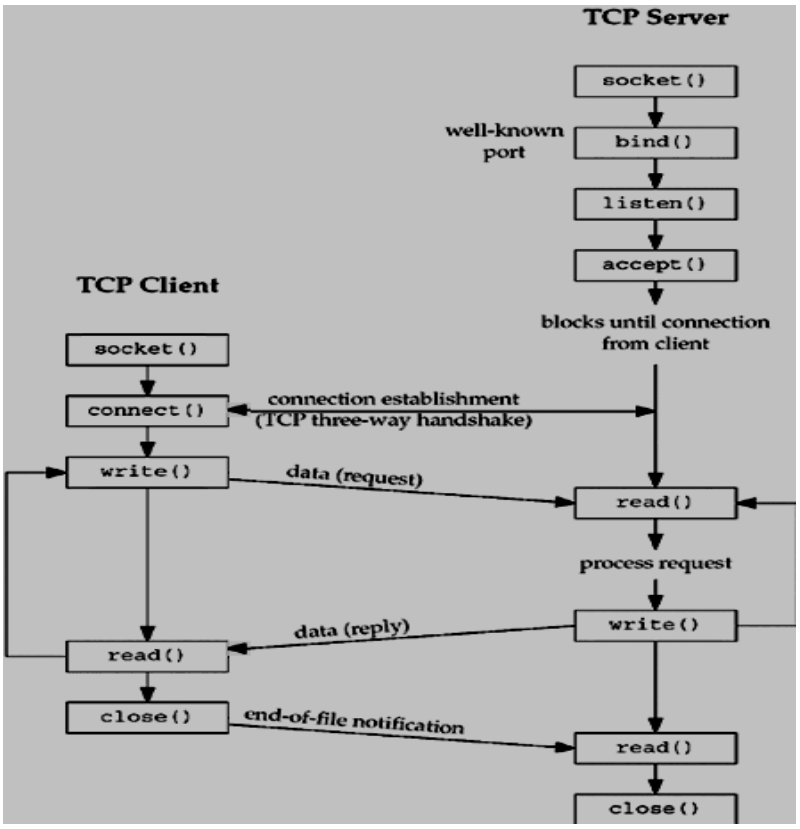


Figura 4 Comunicación socket.

El puerto que utiliza un socket debe ser mayor de 1024 y menor de 49151, pues estos son los llamados puertos registrados y son los de libre utilización.

Para comunicarse con otro proceso usando sockets debe conocerse:

1. Dirección IP (32 bits) de la máquina donde se ejecuta el proceso. Alternativamente su nombre para consultar el servicio de nombre DNS[2] (traducción dominio-IP).
2. Número de puerto que utiliza el proceso en la máquina servidor [8].

[2] DNS: Domain Name System (Resolución de nombres de dominio) es un sistema de nomenclatura jerárquico descentralizado para dispositivos conectados a redes IP como Internet o una red privada.

Existen varios tipos de sockets, los más importantes son:

- Socket de flujo (Stream): utiliza el protocolo de transporte TCP (Transmission Control Protocol o Protocolo de Control de Transmisión) y establece una conexión entre los dos extremos que permite enviar flujos de bytes en ambas direcciones. Es fiable pues se asegura el orden de entrega de los mensajes.
  Existen dos papeles definidos implícitamente:
  -sockets de servidor: esperan recibir conexiones.
  -sockets de tráfico: inician conexiones (en los clientes) y envían/reciben datos (en clientes y servidor).
- Socket de datagrama (Datagram): utiliza protocolo de transporte UDP (User Datagram Protocol) y no se establece conexión, cada datagrama es independiente. No es fiable pues no se asegura el orden en la entrega y mantiene la separación entre mensajes[8].

Ventajas del uso de Sockets:

- Proporcionan una comunicación de dos vías entre dos ordenadores.
- Permite maximizar la flexibilidad a la hora de desarrollar aplicaciones porque se tiene el control total de los datos enviados.
- Tiene gran disponibilidad pues la interfaz Socket está disponible en múltiples sistemas operativos y lenguajes de programación[8].

Desventaja del uso de Sockets:

- Se necesita realizar un manejo explícito de los datos a transmitir y es complejo de implementar[8].

Análisis de la selección del mecanismo a utilizar

Tabla 1 Comparativa entre Socket y Pipes

Aspectos	Sockets	Pipes
Tipo de referencia	Referenciado por descriptores	Referenciado por array de descriptores
Tipo de comunicación	Comunicación entre procesos de la misma y diferentes máquinas	Comunicación solo entre procesos de la misma máquina
Dirección de la comunicación	Comunicación bidireccional	Comunicación unidireccional
Arquitectura	Filosofía cliente-servidor	Simple intercambio de información

Después de un estudio detallado de los mecanismos, el autor decidió usar Socket ya que permite la comunicación entre procesos de la misma y diferentes máquinas, intercambiarse cualquier flujo de datos, generalmente de forma fiable y ordenada. Tiene como ventaja que permite que el enlace entre dos sockets use una comunicación bidireccional que lo diferencia de pipes, que solamente permite comunicación unidireccional entre procesos de la misma máquina. La comunicación entre procesos a través de socket se basa en la filosofía cliente-servidor, un proceso actuará de proceso de servidor cuyo nombre conocerá el proceso cliente, que podrá comunicarse con el proceso servidor a través de la conexión con dicho socket, mientras tanto pipes es simplemente un intercambio de información. Tienen además gran disponibilidad, pues la interfaz Socket está disponible en múltiples sistemas operativos y lenguajes de programación. Entre los tipos de sockets estudiados se determinó utilizar el socket de flujo pues es el más fiable y trabaja sobre el protocolo de transporte TCP/IP.

1.3 Estudio de protocolos

Entre los tipos de protocolo en la comunicación entre ordenadores se tiene

1.3.1 Protocolo de Control de Transmisión/Protocolo de Internet(TCP/IP)

TCP/IP fue desarrollado y demostrado por primera vez en 1972 por el Departamento de Defensa de los Estados Unidos, ejecutándolo en ARPANET, una red de área extensa de dicho departamento. Son los protocolos que se utilizan en Internet para transmitir datos, el TCP está orientado a la conexión que establece una línea de diálogo entre el emisor y el receptor antes de que se transfieran los datos, y el IP trata cada paquete de forma independiente e incluye en la cabecera información adicional para así controlar la información. Estos protocolos garantizan que la comunicación entre dos aplicaciones sea precisa[9].

Se le denomina conjunto de protocolos TCP/IP, en referencia a los dos protocolos más importantes que lo componen, los cuales fueron los primeros en definirse y son los más utilizados. Pero existen más de 100 protocolos diferentes en este conjunto[9].

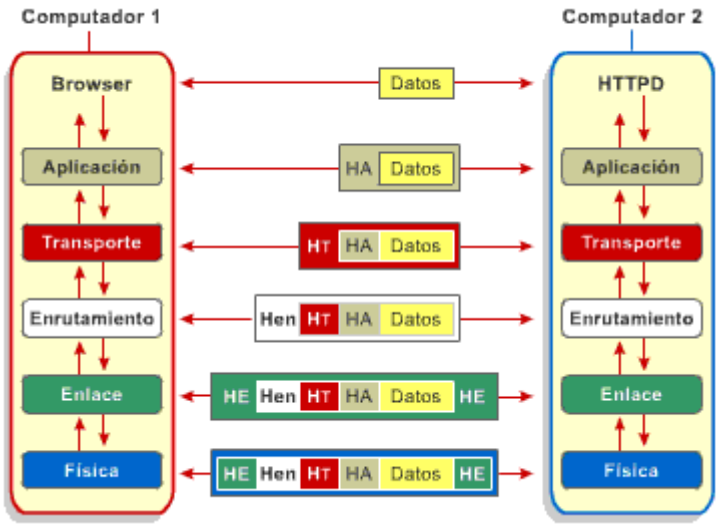


Figura 5 Transmisión de datos TCP-IP.

Ventajas de TCP/IP:

- Consume pocos recursos de red.
- Trabaja sobre una gran variedad de hardware y sistemas operativos.
- Está diseñado para enrutar y tiene un grado muy elevado de fiabilidad.
- Es adecuado para redes grandes y medianas, así como en redes empresariales.
- Es compatible con las herramientas estándar para analizar el funcionamiento de la red.
- Ofrece mayor fiabilidad pues envía información adicional en el paquete para asegurar la conexión permitiendo que los datos vayan correctamente del emisor al receptor, en el orden estipulado, y completos.
- Debe ser utilizado en aplicaciones donde es más importante que los datos lleguen correctamente a que lleguen rápidamente[9].

Desventajas de TCP/IP:

- Es más difícil de configurar y de mantener.
- Al enviar información adicional en el paquete para asegurar la conexión, como el tamaño del paquete es limitado, pierde espacio útil para esta función[9].

1.3.2 Protocolo de Transferencia de Archivos (FTP)

La implementación del FTP se remonta a 1971, cuando se desarrolló un sistema de transferencia de archivos (descrito en RFC141) entre equipos del Instituto Tecnológico de Massachusetts. Es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP, basado en la arquitectura cliente-servidor, pues desde un equipo cliente se puede conectar a un servidor para descargar archivos desde él o para enviarle archivos, independientemente del sistema operativo utilizado en cada equipo[10].

FTP define la manera en que los datos deben ser transferidos a través de una red TCP/IP y este permite:

- La conexión a un sistema remoto.
- Observar los directorios remotos.
- Cambiar de directorio remoto.
- Copiar uno o varios archivos hacia el directorio local.
- Copiar uno o varios archivos hacia el directorio remoto[10].

El protocolo FTP emplea los permisos de ejecución, lectura y escritura debido a que se desarrolló en entornos de tipo UNIX similares a los populares GNU/Linux. Es el más conocido y utilizado para la transferencia de archivos en Internet pues es el ideal para transferir grandes bloques de datos por la red[10].

1.3.3 Llamada a procedimiento remoto (RPC)

El RPC fue inventado por la empresa SUN Microsystems y puede ser usado por medio de los protocolos TCP/IP o UDP/IP. Es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos[11].

Ejemplos de entornos RPC [11]:

- Sun-RPC (ONC-RPC: Open Network Computing-RPC): RPC es muy extendido en entornos UNIX, infraestructura sobre la que se ejecuta NFS (servicio de sistema de ficheros en red), NIS (servicio de directorio).
- DCE/RPC (Distributed Computing Environment RPC): RPC definido por la Open Software Foundation.
- Java-RMI: invocación de métodos remotos en Java.
- CORBA (Common Object Requesting Broker Architecture): soporta la invocación de métodos remotos bajo un paradigma orientado a objetos en diversos lenguajes.
- SOAP (Simple Object Access Protocol): protocolo RPC basado en el intercambio de datos (parámetros + resultados) en formato XML (eXtensible Markup Language o Lenguaje de Marcado Extensible).
- DCOM (Distributed Component Object Model): Modelo de Objetos de Componentes Distribuidos de Microsoft, con elementos de DCE/RPC.
- .NET Remoting: infraestructura de invocación remota de .NET.

Ventajas de RPC:

- El protocolo es un gran avance sobre los sockets usados hasta el momento.
- El programador no tiene que estar pendiente de las comunicaciones, estando estas encapsuladas dentro de las RPC, pues el objetivo del mismo es ocultar/abstraer los detalles relativos a la comunicación que son comunes a diversas aplicaciones.
- Las RPC son muy utilizadas dentro del paradigma cliente-servidor[11].

Desventaja de RPC:

- Las comunicaciones RPC se basan en la idea de que el receptor está operativo para poder invocar una cierta función, no se puede suponer que el receptor siempre estará operativo y esperando a comunicarse[11].

1.3.4 XML-RPC

Es un protocolo de llamada a procedimiento remoto que usa XML para codificar los datos y HTTP como protocolo de transmisión de mensajes. Fue creado por la empresa UserLand Software en asociación con Microsoft en el año 1998. XML-RPC utiliza las peticiones POST de HTTP(Hypertext Transfer Protocol o protocolo de transferencia de hipertexto) para enviar un mensaje, en formato XML, señalando el procedimiento que se va a ejecutar en el servidor y los parámetros, y el servidor devuelve el resultado en formato XML[12].

Ventajas del XML-RPC:

- Es un protocolo muy simple ya que sólo define unos cuantos tipos de datos y comandos útiles, además de una descripción completa de corta extensión.
- La simplicidad del XML-RPC está en contraste con la mayoría de los protocolos RPC que tiene una documentación extensa y requiere considerable soporte de software para su uso[12].

Desventaja del XML-RPC:

- Funciona sobre Internet.

1.4 Estudio de los frameworks para el desarrollo de aplicaciones con comunicación socket

El análisis de algunos framework cliente-servidor NIO[3], permitió recopilar la información y así sintetizar los conocimientos necesarios para proponer una solución correcta, centrando la investigación en aquellos que basan su funcionamiento en socket y se encuentren implementados bajo el lenguaje Java.

1.4.1 Netty

El autor David Wheeler[4] plantea que *"todos los problemas en la informática pueden ser resueltos por otro nivel de indirección"*. Como un marco cliente-servidor NIO, Netty ofrece un nivel de indirección. Netty simplifica la programación de red de los servidores TCP o UDP, pero todavía puede acceder y utilizar las API(Application Programming Interface o interfaz de

[3] **NIO (Non-sequential I/O)**, es una forma de procesamiento de entrada / salida que permite continuar el procesamiento antes de que la transmisión haya finalizado.

[4] **David John Wheeler**, miembro de la Royal Society (9 de febrero de 1927 – 13 de diciembre de 2004) fue un científico de computación. Nació en Birmingham y se graduó en 1928 en el Trinity College, Cambridge.

programación de aplicaciones) de bajo nivel porque Netty proporciona abstracciones de alto nivel[13].

Netty es un framework cliente-servidor NIO, que permite el desarrollo rápido y fácil de aplicaciones de red, como servidores de protocolo y clientes. Netty le ofrece una nueva forma de desarrollar sus aplicaciones de red, lo que lo hace fácil y escalable. Esto se logra mediante la abstracción de la complejidad que implica distancia y proporcionando una API fácil de usar que desacopla de lógica de negocio a partir del código de manejo de red. Debido a que está construido para NIO, toda la API de Netty es asíncrona [14].

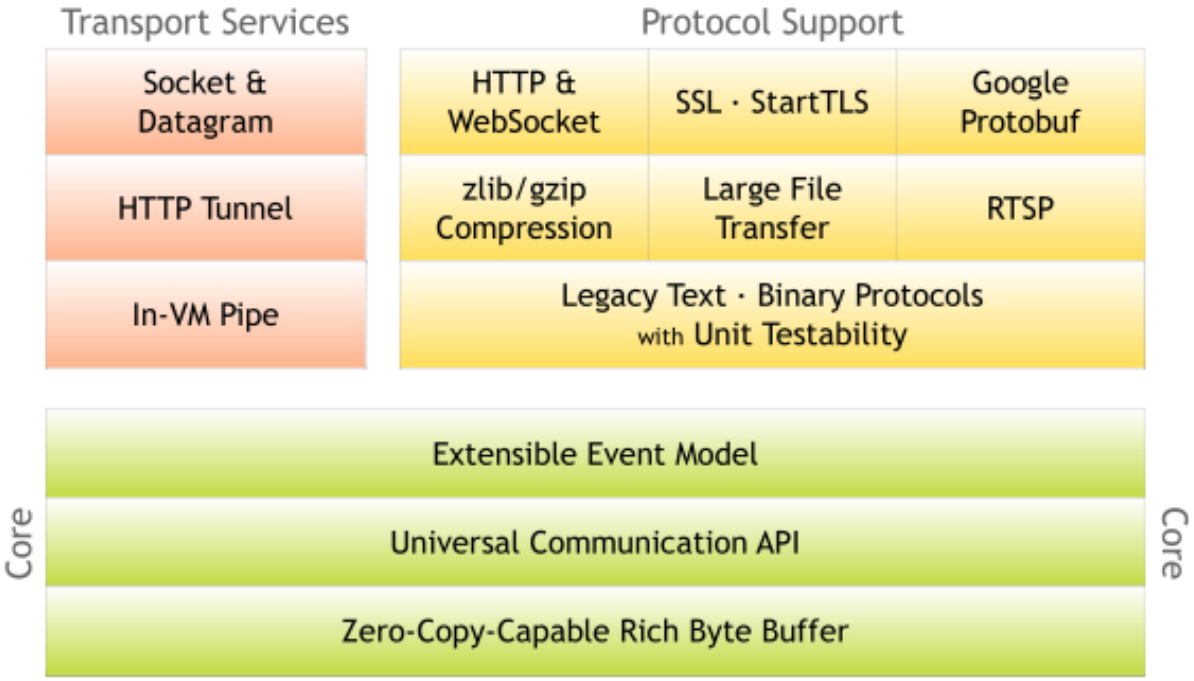


Figura 6 Componentes de Netty.

Características de Netty:

- **Facilidad de uso:**
    1. Javadoc bien documentado y muchos ejemplos proporcionados.
- No hay dependencias adicionales excepto JDK 1.6 (o superior). Algunas funciones sólo se admiten en Java 1.7 o superior.

- **Diseño:**

    1. API unificada para varios tipos de transporte.
    2. Flexible de usar.
    3. Modelo de hilo simple pero potente.
    4. Soporte de socket de datagrama auténtico sin conexión.
    5. Encadenamiento de lógicas para facilitar la reutilización.

- **Rendimiento:**

    1. Mejor rendimiento; menor latencia que las API de Java principales.
    2. Menor consumo de recursos debido a la agrupación y reutilización.
    3. Copia de memoria minimizada innecesaria[13].

1.4.2 Apache Mina

Apache MINA es un framework de aplicaciones de red que ayuda a los usuarios a desarrollar aplicaciones de red de alto rendimiento y alta escalabilidad fácilmente. Proporciona una API asíncrona abstracta basada en eventos a través de varios transportes como TCP / IP y UDP / IP a través de Java NIO.

Apache MINA se llama a menudo:

- NIO framework-library
- Client-server framework-library
- networking-socket library[15].

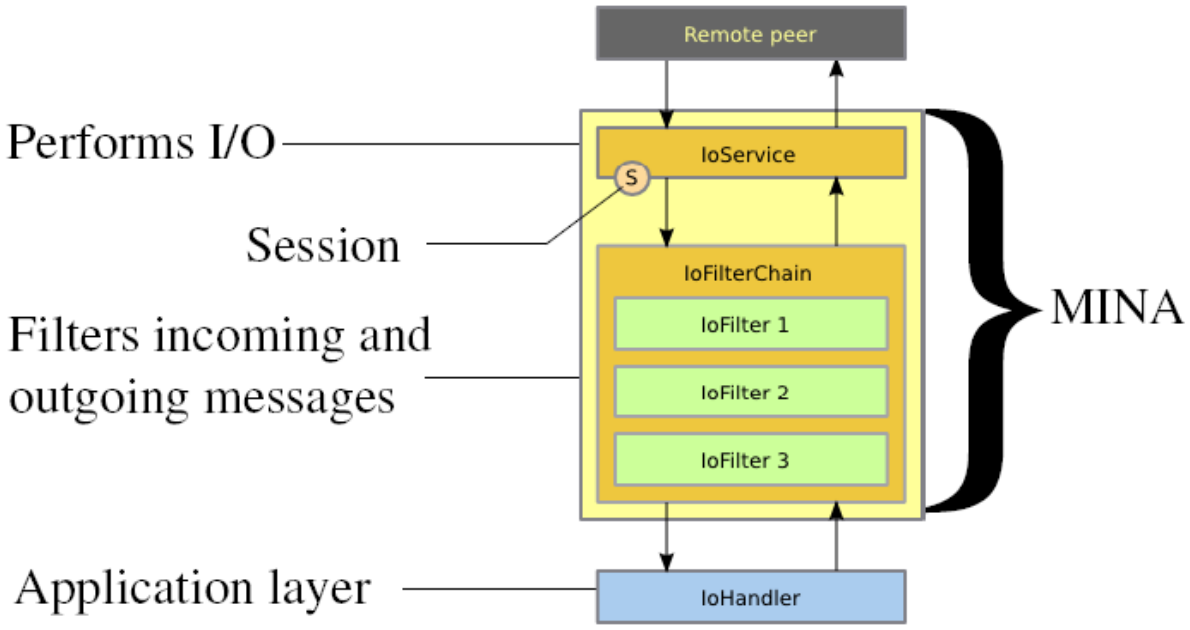


Figura 7 Componentes de Apache Mina.

Características de Apache Mina:

- **Conservable y reutilizable:**
    1. Motor de red - Servicio de E / S MINA.
    2. Códec de protocolo - Marco de códec MINA.
- **Extensible:**
    1. Modificación en tiempo de ejecución del comportamiento de la aplicación mediante filtros

Análisis de la selección de lo frameworks

A partir de lo anteriormente expuesto el autor decide utilizar Netty ya que cuenta con una buena documentación para su uso en aplicaciones y diversos ejemplos de su implementación en distintos escenarios además que es muy flexible de usar. Cuenta con mejor rendimiento y una menor latencia que las demás API de Java, además que tiene un menor consumo de los recursos debido a la agrupación y reutilización.

1.5 Herramientas, lenguajes y metodología a utilizar

Para la determinación de las herramientas y lenguajes que se utilizaron en el desarrollo de la solución, se tuvo en cuenta el análisis de las utilizadas durante el desarrollo del software REKO. Las mismas se mantendrán en la implementación de la solución de la presente investigación, debido a que permite disminuir la curva de aprendizaje y el tiempo empleado en el desarrollo de la solución.

1.5.1 Metodología de desarrollo

Las metodologías para el desarrollo del software imponen un proceso disciplinado con el fin de hacerlo más predecible y eficiente. Una metodología define una representación que permite facilitar la manipulación de modelos, y la comunicación e intercambio de información entre todas las partes involucradas en la construcción de un sistema.

El ciclo de vida del proceso de desarrollo de software de la solución estará guiado por la metodología Proceso Unificado Ágil (AUP, Agile Unified Process, por sus siglas en inglés) debido a que es la designada por la universidad para la actividad productiva, específicamente para los proyectos de desarrollo[16].

Metodología AUP

Describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP[17].

Ha sido adaptada a las necesidades de desarrollo de software en la Universidad de las Ciencias informáticas. En la variación efectuada se proponen 3 fases para guiar el proceso de desarrollo de los proyectos en la universidad:

Inicio: durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.

Ejecución: en esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente. Además, en la transición se capacita a

los usuarios finales sobre la utilización del software.

Cierre: en esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

1.5.2 Herramientas y lenguajes:

A continuación, se especifican las herramientas que por sus características fueron seleccionadas para su uso en la elaboración de la solución.

Visual Paradigm para UML 8.0: herramienta multiplataforma que posee licencia gratuita y comercial diseñada para soportar el ciclo de vida completo del proceso de desarrollo del software, a través de la representación de todo tipo de diagramas, con capacidad de ingeniería directa e inversa enfocada al negocio que genera un software de mayor calidad.

Java: es un lenguaje de programación de alto nivel que aplica un modelo de programación orientada a objetos, que se puede utilizar para crear aplicaciones completas que pueden ejecutarse en un único ordenador o ser distribuidos entre servidores y clientes en una red[18].

UML 2.0: es un lenguaje de modelado estándar para el desarrollo de sistemas de software orientado a partir del paradigma de programación orientada a objetos, permitiendo especificar, visualizar, construir y documentar todos los elementos que conforman dichos sistemas[19].

Spring 4.2.2: contenedor y framework de peso ligero, basado en inyección de dependencias, promoviendo el bajo acoplamiento pues los objetos reciben pasivamente sus dependencias en lugar de crearlas o buscar los objetos dependientes por sí mismos. Brinda facilidades para desarrollar una aplicación empresarial en JEE[20].

Intellij Idea Community 2016.3.3: IntelliJ IDEA Community Edition es la versión de código abierto de IntelliJ IDEA, un IDE de primer nivel para Java, Groovy y otros lenguajes de programación como Scala o Clojure[21].

Netty 4.0.4: es una librería/framework NIO basada en eventos para el desarrollo de aplicaciones con comunicación Socket[22].

1.6 Consideraciones parciales del capítulo

- Para la comprensión del trabajo se profundizó en el conocimiento de algunos conceptos fundamentales con lo que se logró realizar un esbozo de las principales características y conceptos relacionados con los mecanismos de comunicación entre ordenadores.
- Se definieron y justificaron las herramientas, metodología y lenguajes a utilizar para dar solución al problema planteado.

Capítulo 2. Características del sistema

El presente capítulo expone las características del sistema y se plantean, el modelo de dominio y los requisitos funcionales y no funcionales del mecanismo de comunicación. Además, se realiza la definición de las Historias de Usuario.

2.1 Funcionamiento del software de replicación REKO

Cada instancia del mecanismo se identifica como un nodo con un identificador único. Los nodos pueden estar agrupados en etiquetas. Desde cada nodo puede configurarse la réplica hacia otro nodo o hacia una etiqueta. A partir de un nodo origen o fuente pueden realizarse distintas configuraciones según los nodos y/o etiquetas destino que se definan. Pudiendo definir configuraciones de réplica para nodos y/o etiquetas. Por cada configuración, se registran las configuraciones independientes de las tablas que se desean replicar. Cada tabla puede ser configurada para replicar según la acción que se efectúe sobre ella: inserción, actualización y/o eliminación. Además, pueden definirse filtros SQL para determinar por cada acción si se replicará o no el cambio. Cada cierto intervalo de tiempo (definidos por el administrador de réplica), el Capturador de Cambios de REKO registra los cambios realizados sobre la base de datos y crea un grupo replicable, que contiene un grupo de acciones replicables, cada una de las cuales almacena toda la información necesaria para replicar los cambios. A partir de estas entidades, las configuraciones registradas y los filtros asociados a estas, se determinan los destinos hacia donde se enviará el grupo replicable. De la propagación de los grupos replicables se encarga el Distribuidor de Datos empleando el servidor de mensajería Java Apache ActiveMQ. En cada nodo destino, una vez recibido un grupo replicable, pasa al Aplicador de Cambios para ejecutar en la base de datos del destino las acciones que contiene. Existen mecanismos de chequeo de envío, confirmación de recepción y registro en una base de datos local, de los grupos replicables; los que garantizan la integridad y persistencia de los datos que se replican ante posibles eventualidades.

2.2 Propuesta del sistema

Para brindar una solución a la problemática planteada se propuso que, a partir de una configuración de réplica de fichero, la cual contiene el o los directorios que serán replicados, el sistema crea para cada uno de los ficheros contenidos un capturador de eventos, el cual se encarga cuando ocurren modificaciones en los archivos (creación, modificación) de informar al sistema. Cuando se informa la existencia de algún cambio, el sistema carga el fichero y este es enviado hacia el nodo remoto a través del servidor de ficheros, el cual se encarga mediante

mensajes y con el uso de canales, de establecer la comunicación entre los nodos que intervienen en el proceso de réplica. Una vez recibido el fichero por el nodo destino, este realiza su escritura en el sistema de archivos teniendo en cuenta la dirección que tenía en el nodo origen. Terminado el proceso de réplica de ficheros se informa mediante una notificación el estado de la operación.

2.3 Modelo de dominio

Un modelo de dominio es una representación visual de las clases conceptuales u objetos del mundo real que se definen en un dominio de interés, permitiendo establecer un lenguaje común entre los usuarios, clientes, desarrolladores y demás interesados[23].

La realización de este modelo permite además delimitar el alcance del domino del problema, añadir precisión y enfoque para la discusión entre el equipo de técnicos y de negocio.

Diagrama de clases del modelo de dominio

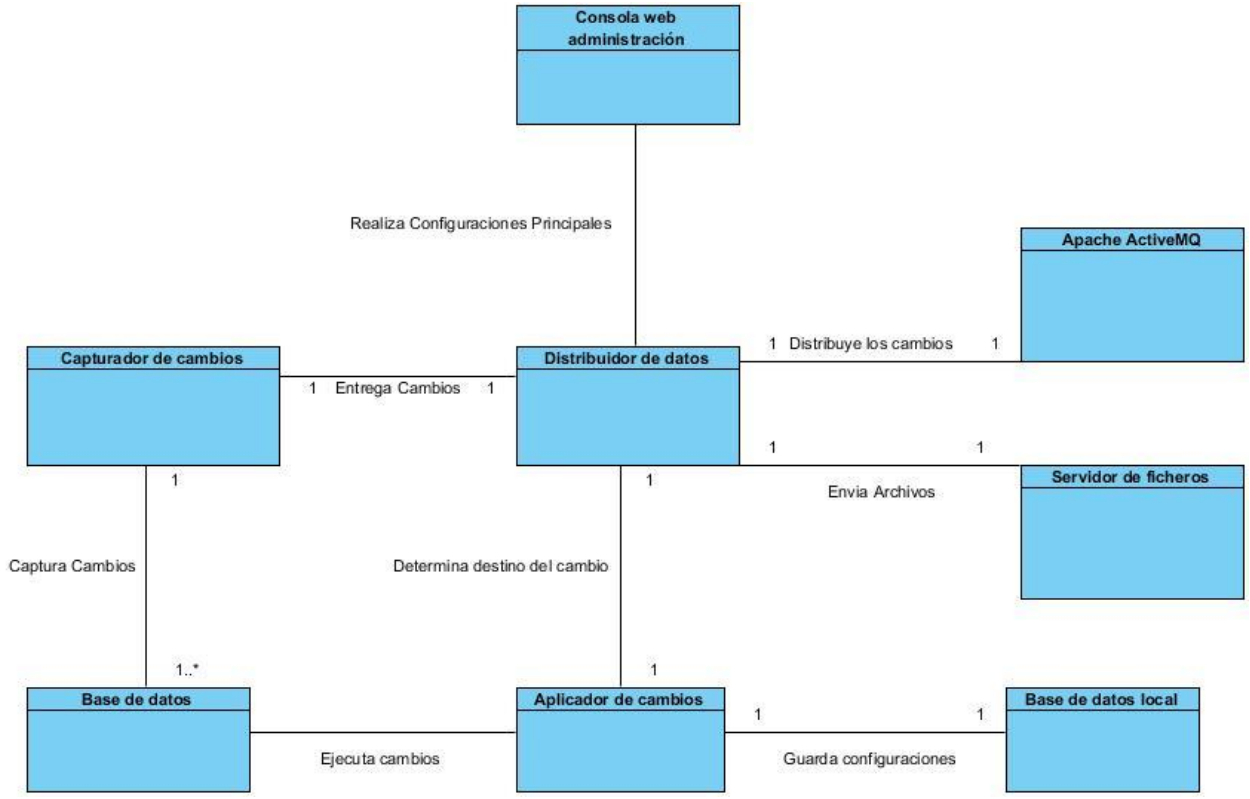


Figura 8 Diagrama de clases del dominio.

En la figura 8 se muestra el diagrama de clases del dominio del replicador Reko con el Servidor de ficheros incluido, que es la solución propuesta en esta investigación.

Descripción de las clases del modelo de dominio:

Capturador de cambios: captura los cambios que se realizan sobre la Base de datos (BD) y se los entrega al Distribuidor de Cambios.

Distribuidor de cambios: determina para dónde debe ser enviado cada cambio realizado en la BD, los envía y se responsabiliza de que cada cambio llegue a su destino.

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

BD local: es utilizada para guardar las configuraciones propias de la réplica, las acciones al aplicarse sobre la BD dado un conflicto y las acciones o transacciones que no han podido llegar a su destino.

Consola web de administración: representa la interfaz del replicador. 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 sistema.

Servidor apache ActiveMQ: servidor de mensajería bajo la especificación *Java Message Service(JMS)*, es utilizado como punto intermedio en la distribución de la información.

Servidor de ficheros: se encarga de realizar la comunicación entre instancias de REKO durante el envío de ficheros de gran tamaño.

Base de datos: es la BD 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.

2.4 Especificación de requisitos de software

Según la IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) “es un conjunto de recomendaciones para la especificación de los requerimientos o requisitos de software el cual tiene como producto final la documentación de los acuerdos entre el cliente y el grupo de desarrollo para así cumplir con la totalidad de exigencias estipuladas” [24].

Requisitos funcionales

Describen como debe comportarse el sistema ante un determinado estímulo. Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares en distintos tipos de situaciones específicas [25].

Los requisitos funcionales identificados se mencionan a continuación:

RF1: Configurar conexión del servidor de ficheros.

RF2: Configurar réplica de ficheros.

RF3: Guardar configuración de réplica de ficheros.

RF4: Capturar cambios en el fichero.

RF5: Enviar ficheros.

RF5.1: Capturar fichero con cambios.

RF5.2: Persistir fichero con cambios.

RF5.3: Enviar fichero con cambios.

RF6: Recibir fichero.

- **RF6.1:** Recibir fichero con cambios.
- **RF6.2:** Escribir fichero en el sistema de archivos.

RF7: Enviar notificaciones

Requisitos no funcionales

Los **requerimientos no funcionales** representan características generales y restricciones de la aplicación o sistema que se esté desarrollando.
Suelen presentar dificultades en su definición dado que su conformidad o no conformidad podría ser sujeto de libre interpretación, por lo cual es recomendable acompañar su definición con criterios de aceptación que se puedan medir.
Entre los ejemplos de requerimientos no funcionales presentados, tenemos los referidos a atributos como la mantenibilidad, portabilidad, seguridad y adecuación funcional del sistema. También presentamos ejemplos de requerimientos no funcionales organizacionales y externos[26].

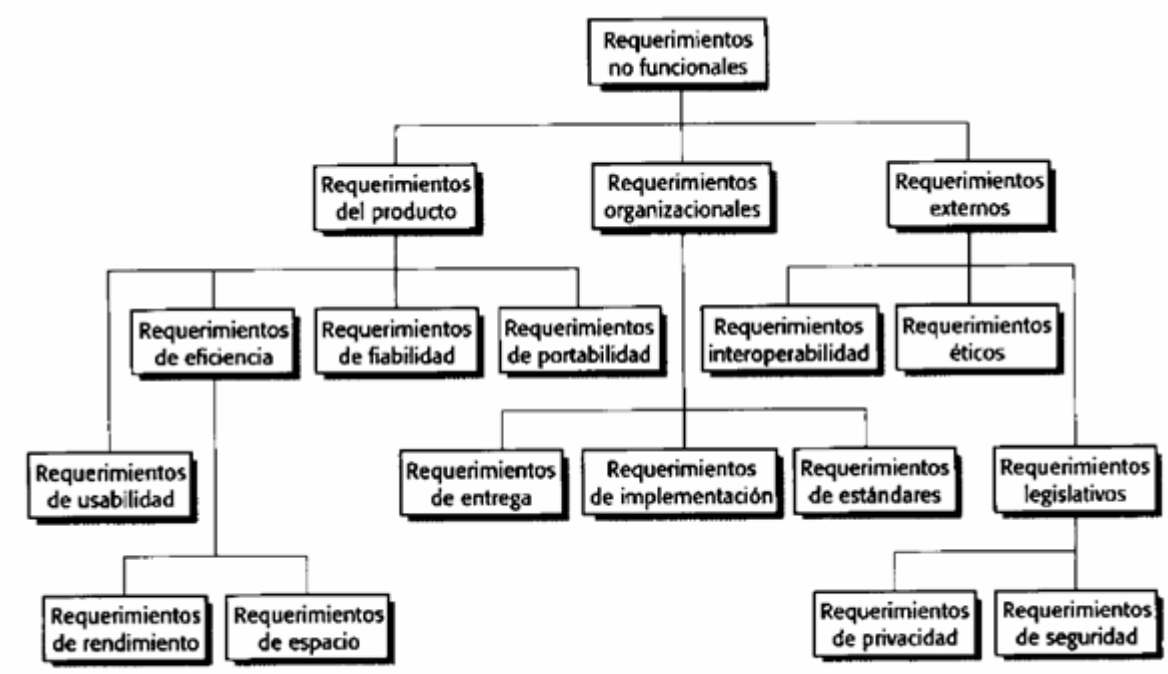


Figura 9 Clasificación de requisitos no funcionales.

Tabla 2 Requisitos no funcionales

Atributo de Calidad	Mantenibilidad.
Sub-atributos/Sub-característica	Capacidad para ser probado.
Objetivo	Facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios.
Origen	Arquitecto de software y el analista
Artefacto	Diseños de casos de pruebas.
Entorno	El sistema desplegado.
Estímulo	**Respuesta: Flujo de eventos (Escenarios)**
<<1>>. <<a>>< Facilidad con la que se pueden establecer criterios de prueba para el sistema>	

Se definen y delimitan las funciones según los requisitos funcionales a implementar.	1. El sistema responde de manera satisfactoria en todos los escenarios de prueba a que fue sometido, validando las variables de entrada y salida que fueron especificadas.
Medida de respuesta	
Escenario de prueba del sistema.	

Tabla 3 Requisitos no funcionales

Atributo de Calidad	Portabilidad.
Sub-atributos/Sub-característica	Adaptabilidad.
Objetivo	Capacidad del producto que le permite ser adaptado de forma efectiva y eficiente a diferentes entornos determinados de hardware, software, operacionales o de uso.
Origen	Arquitecto de software.
Artefacto	El sistema.
Entorno	El sistema desplegado.
Estímulo	**Respuesta: Flujo de eventos (Escenarios)**
<<1>>. <<a>>< Capacidad del sistema de adaptarse de forma efectiva a diferentes entornos>	
El sistema está diseñado con tecnologías que permiten que este se adapte a cualquier sistema operativo.	1. El sistema posee un *script* de inicio que permite personalizar los recursos de RAM de uso de la aplicación. 2. El sistema posee un fichero de configuración permite configurar los parámetros referente a las prestaciones que posea la estación de trabajo donde se encuentre instalado.

Medida de respuesta
El ambiente de despliegue del sistema.

Tabla 4 Requisitos no funcionales

Atributo de Calidad	Adecuación funcional.
Sub-atributos/Sub-característica	Integridad funcional.
Objetivo	Grado en el que el conjunto de funciones cubre todas las tareas y objetivos del usuario especificados.
Origen	Proveedor de requisitos.
Artefacto	El sistema.
Entorno	El sistema desplegado.
Estímulo	**Respuesta: Flujo de eventos (Escenarios)**
<<1>>. <<a>>< Grado en el que el sistema cubre todas las tareas y objetivos especificados>	
El desarrollo del sistema esta guiado por las necesidades expresadas por parte de los proveedores a través de las historias de usuario, dándole total cumplimiento a sus requisitos planteados.	1. El sistema brinda todas las funciones que se contemplaron en los requisitos planteados.
Medida de respuesta	
Analizar las funcionalidades del sistema.	

Tabla 5 Requisitos no funcionales

Atributo de Calidad	Seguridad.
Sub-atributos/Sub-característica	No repudio.
Objetivo	Grado en que las acciones o eventos pueden ser probados a haber tenido lugar, por lo que los eventos o acciones no pueden ser repudiados más tarde.
Origen	Arquitecto de software.
Artefacto	El sistema.
Entorno	El sistema desplegado.
Estímulo	**Respuesta: Flujo de eventos (Escenarios)**
<<1>>. <<a>>< Sistema de eventos o trazas.>	
El sistema se inicia.	1. El sistema crea un conjunto de trazas a nivel de ficheros en la raíz de la solución en una carpeta nombrada log donde se almacenan tres archivos: • Audit.log: contiene todas las trazas relacionadas con las respuestas de la aplicación al cliente. • Error.log: contiene los errores ocurridos en el sistema • System.log: contiene todas las trazas relacionadas con las acciones ejecutadas por el sistema. Nota: Este sistema de trazas garantiza el no repudio sobre las acciones realizadas en el sistema.

Medida de respuesta
Navegar en el sistema.

2.5 Historias de Usuarios

Las Historias de Usuarios (HU) son descripciones cortas y simples de las funcionalidades del sistema, narradas desde la perspectiva de la persona que desea dicha funcionalidad, poseen una descripción escrita que será utilizada para planificar y posteriormente disgregar los detalles con el dueño del producto[27].

A continuación, se muestran las Historias de Usuario de los RF1, RF2, RF6 y RF7, las restantes en el Anexo 1.

Tabla 6 Historia de Usuario RF1

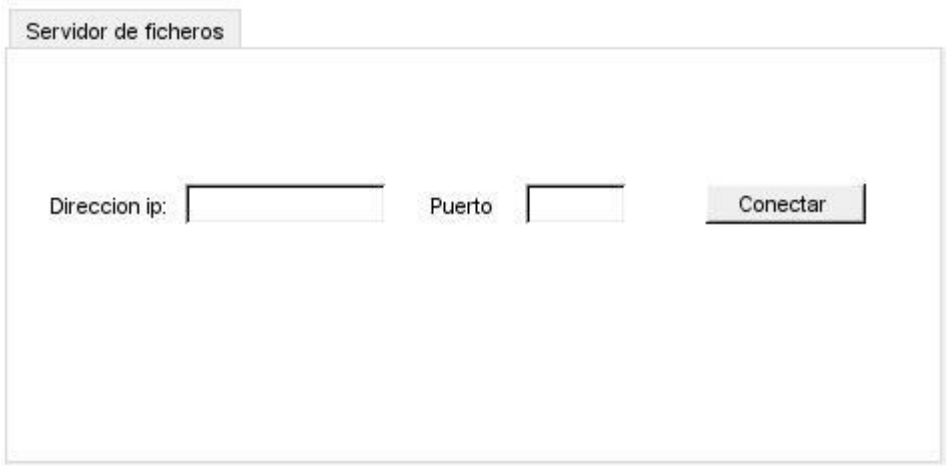
Historia de Usuario	
Número: HU1	**Nombre del requisito :** Configurar conexión de servidor de ficheros
Programador: Rogelio González Martínez	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo estimado:** 2 semanas
Riesgo en desarrollo: Bajo	**Tiempo real:** 90 horas
Descripción: El sistema debe mostrar al usuario en la vista de configuración general la opción ”Servidor de Ficheros” donde se debe introducir la dirección IP del servidor y un puerto para conectarse al servidor de ficheros.	
Observaciones:	
Prototipo de interfaz:	
	

Tabla 7 Historia de Usuario RF2

Historia de Usuario	
Número: HU2	**Nombre del requisito:** Configurar réplica de ficheros
Programador: Rogelio González Martínez	**Iteración asignada:** 1
Prioridad: Media	**Tiempo estimado:** 2 semanas
Riesgo en desarrollo: Bajo	**Tiempo real:** 10horas
Descripción: El sistema debe mostrar en el módulo “Configuración de réplica” en la sección Nueva configuración de réplica una interfaz para configurar la réplica, en este se debe permitir seleccionar las columnas a referenciar y darle la ruta del directorio que se va a marcar como referencia.	
Observaciones:	
Prototipo de interfaz:	

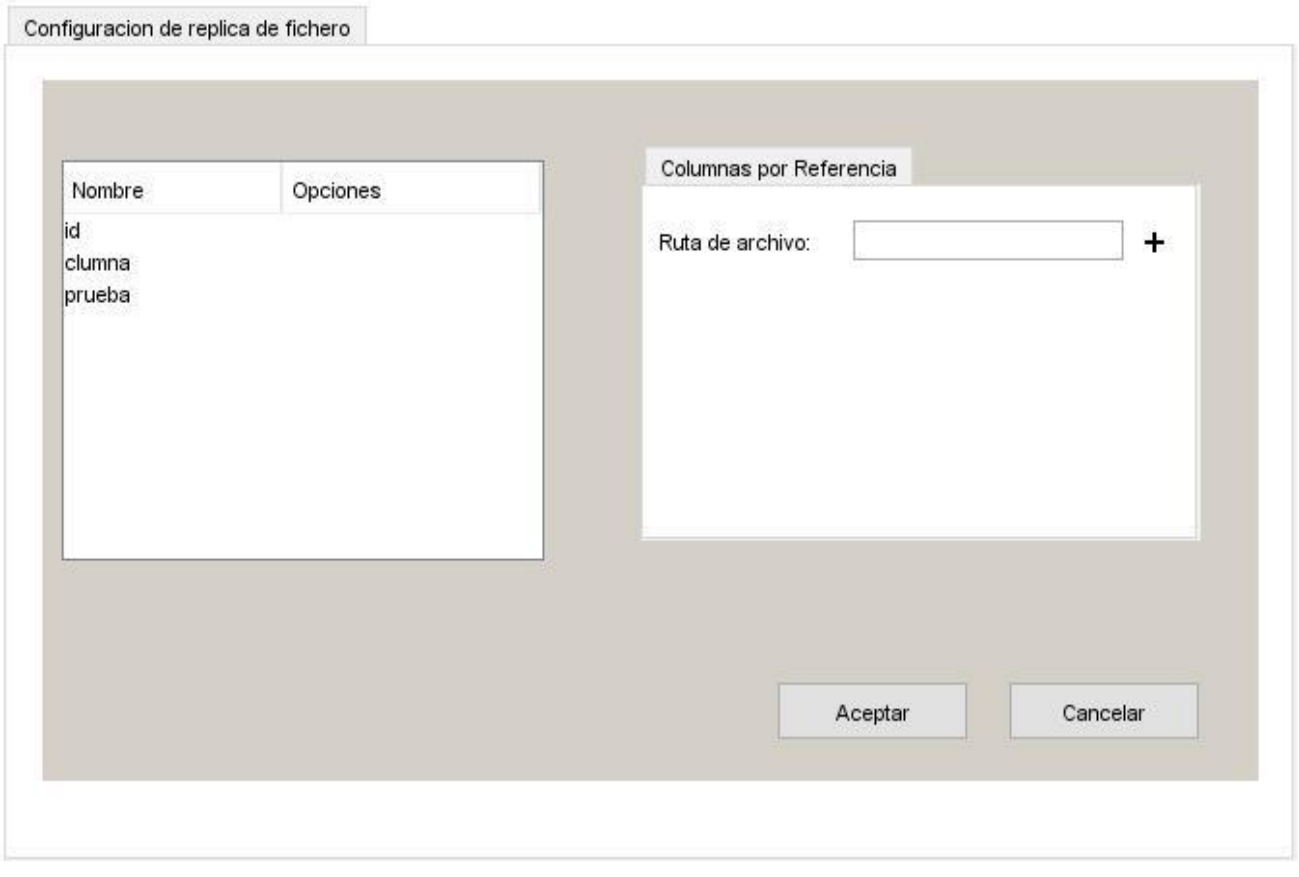


Tabla 8 Historia de Usuario RF6

Historia de Usuario	
Número: HU6	**Nombre del requisito:** Recibir fichero
Programador: Rogelio González Martínez	**Iteración asignada:** 1
Prioridad: Alta	**Tiempo estimado:** 2 semanas
Riesgo en desarrollo: Alto	**Tiempo real:** 90 horas
Descripción: El nodo destino (cliente) deberá recibir los datos enviados desde el servidor de ficheros que es el encargado de transmitirlos desde el nodo origen (cliente), de esto se encarga la clase ***MessageDecodeClient*** que es la encomendada de recibir los arreglos de bytes y con estos crear el archivo en el sistema de archivos y guardarlo.	
Observaciones :	

Tabla 9 Historia de Usuario RF7

Historia de Usuario	
Número: HU7	**Nombre del requisito :** Enviar notificaciones
Programador: Rogelio González Martínez	**Iteración asignada:** 1
Prioridad: Alta	**Tiempo estimado:** 2 semanas
Riesgo en desarrollo: Alto	**Tiempo real:** 90 horas
Descripción: El sistema deberá enviar notificaciones de recibido el archivo y guardado correctamente cuando se termine la réplica del fichero.	
Observaciones:	

2.6 Descripción de la arquitectura

La *"Arquitectura de Software es a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema" [28].*

Por otra parte, la IEEE 1471-2000 define *"La Arquitectura de Software es 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"[28].*

A partir de lo anteriormente planteado se puede concluir que la arquitectura es una vista estructural de alto nivel del sistema que define estilo o combinación de estilos para dar una mejor solución. Se concentra más bien en requerimientos no funcionales ya que los funcionales se resuelven mediante modelado y diseño de la aplicación. La arquitectura es indispensable para el éxito o fracaso de un proyecto.

Para una buena definición de una arquitectura se hace necesario el uso de patrones. *"Cada patrón describe un problema que ocurre infinidad de veces en nuestro entorno, así como la solución al mismo, de tal modo que podemos utilizar esta solución un millón de veces más adelante sin tener que volver a pensarla otra vez*"[29].

Entre los principales objetivos de los patrones de diseño se encuentran:

- Brindar elementos reusables en el diseño de software.
- Evitar tratar de solucionar nuevamente un problema ya conocido o previamente solucionado.
- Estandarizar el modo en que se realiza el diseño de software.

Existen tres categorías de patrones:

- De arquitectura, que define la organización estructural, por ejemplo, el patrón MVC.
- De diseño, que expresa esquemas para definir estructuras de diseño.
- Dialectos, son patrones específicos para un lenguaje de programación[30].

El replicador de datos REKO define su arquitectura basada en n-capas ya que sus partes están separadas por niveles.

La programación por capas es una arquitectura cliente-servidor en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario. La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado. Un buen ejemplo de este método de programación sería el modelo de interconexión de sistemas abiertos[31].

En el diseño de sistemas informáticos actual se suelen usar las arquitecturas multinivel o programación por capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo

que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

El diseño más utilizado actualmente es el diseño en tres niveles (o en tres capas)[31].

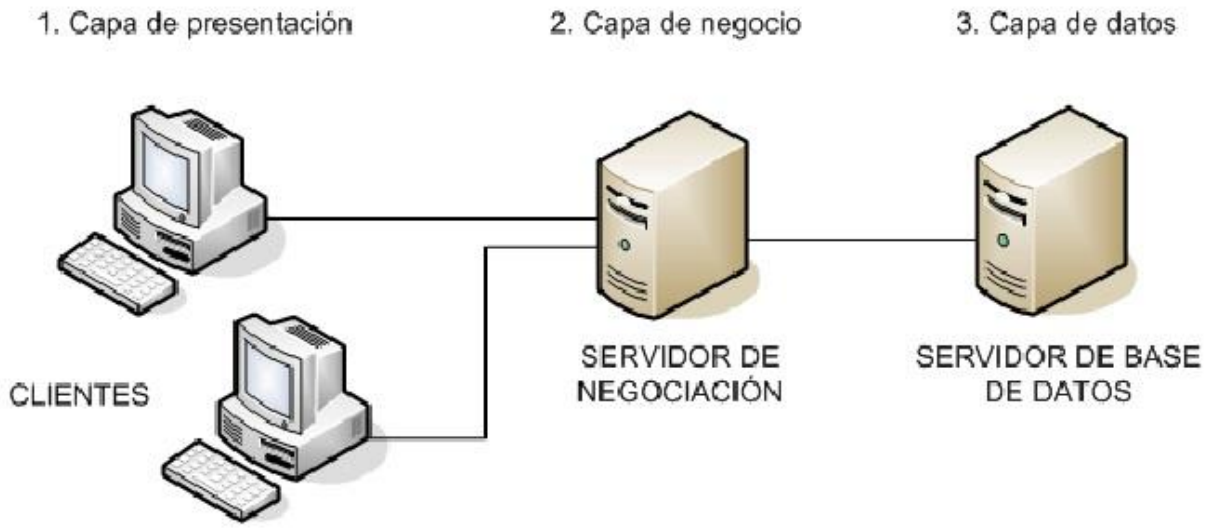


Figura 10 Ejemplo de N-capas

Arquitectura del sistema

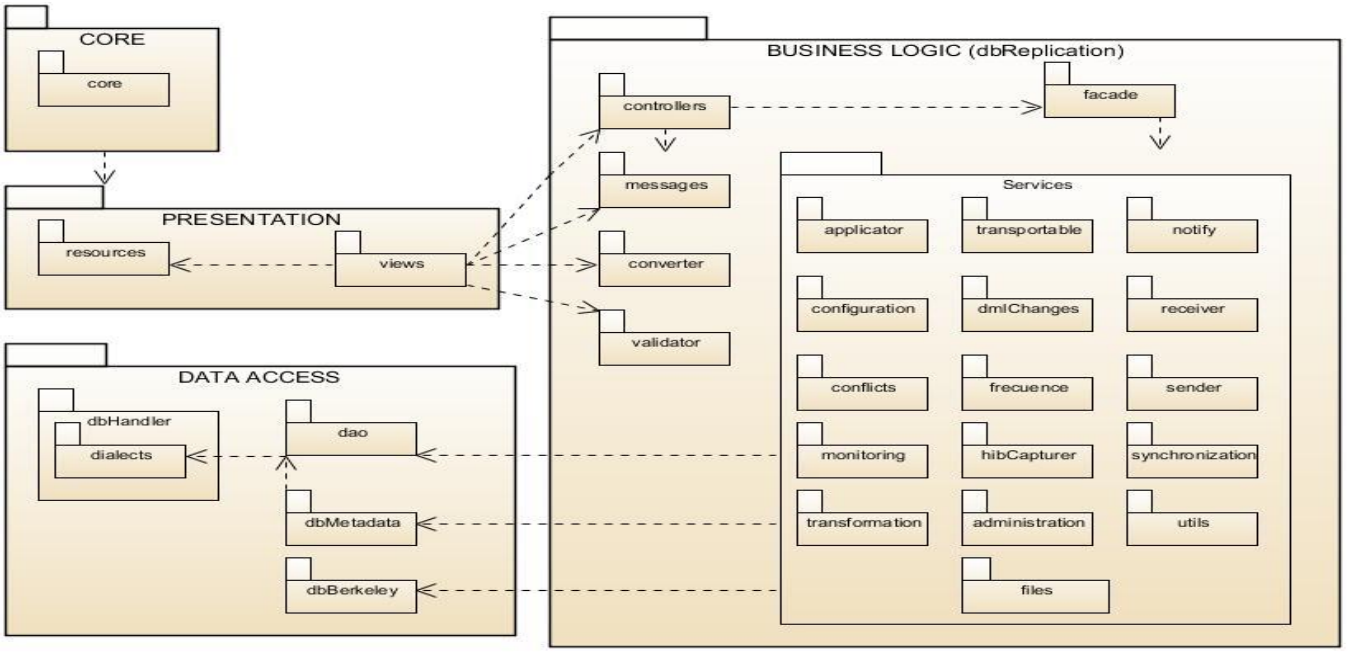


Figura 11 Arquitectura del sistema

Se puede apreciar las distintas capas que componen a REKO separando la vista que está en la capa Presentation, la capa de la lógica del negocio (Bussiness Logic) y la capa de acceso a datos (Data Acces).

2.7 Patrones de Diseño

Los patrones de diseño son situaciones o problemas observados de manera recurrente, y asociados a ellas, una solución óptima (o al menos sumamente efectiva).
Para dar solución al problema planteado se hizo uso de patrones, ejemplificando su implementación en el Capítulo 3.

Patrones GRASP: (se denominan "Patrones GRASP" por sus siglas en inglés: General Responsibility Assignment Software Patterns o Patrones Generales de Software para Asignación de Responsabilidades) Como su nombre lo indica, estos patrones nos indican cual es la manera de asignar responsabilidades a objetos software[32].

- **Singlenton:** Se trata de uno de los patrones más usados y conocidos por los desarrolladores, y también es uno de los patrones más controvertidos. El patrón Singleton se encarga de controlar que únicamente se crea una instancia de una clase en toda la aplicación mediante el uso de un único punto de acceso[32].
- **Experto en información**: el patrón "experto en información" nos sugiere que debemos asignar las responsabilidades a aquellos objetos (o clases de objetos) que disponen de la información para hacerlo[32].
- **Creador:** el patrón "Creador" nos invita a discutir quien es el encargado (o quien debería tener la responsabilidad) de crear un determinado objeto. El patrón creador sugiere encontrar clases de objetos que estén vinculadas (o que se conozcan) para hacerlas responsables de la creación de los objetos, manteniendo así un bajo acoplamiento[32].
- **Alta cohesión:** la cohesión se refiere al grado o la fuerza con que se relacionan algunos elementos. Un elemento con alta cohesión, realiza tareas relacionadas entre sí. Básicamente esto es "Los objetos deben realizar tareas coherentes y relacionadas entre sí"[32].
- **Bajo acoplamiento:** cuando se habla de acoplamiento entre objetos, se hace referencia a la "fuerza" con la que ciertos objetos están relacionados, o dependen unos de otros. Mientras más dependencias tenga un objeto de otros para llevar a cabo sus tareas, más fuerte será el acoplamiento[32].

Patrones GOF(Gang of four o Pandilla de cuatro):

Comportamiento: Los patrones de comportamiento ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos[33].

- **Chain of Responsibility** (Cadena de responsabilidad): Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada[33].
- **Command** (Orden): Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma[33].

2.8 Modelo de diseño

El modelo de diseño es un refinamiento y formalización adicional del modelo del análisis, donde se toman en cuenta las consecuencias del ambiente de implementación. El resultado del modelo de diseño son especificaciones muy detalladas de todos los objetos, incluyendo sus operaciones y atributos. El modelo de diseño se basa en el diseño por responsabilidades[34].

2.8.1 Diagramas de paquetes

Un paquete es un mecanismo utilizado para agrupar elementos de UML. Permite organizar los elementos modelados con UML, facilitando de ésta forma el manejo de los modelos de un sistema complejo. Define un espacio de nombres: dos elementos de UML pueden tener el mismo nombre, con tal y estén en paquetes distintos. Permiten dividir un modelo para agrupar y encapsular sus elementos en unidades lógicas individuales[35].

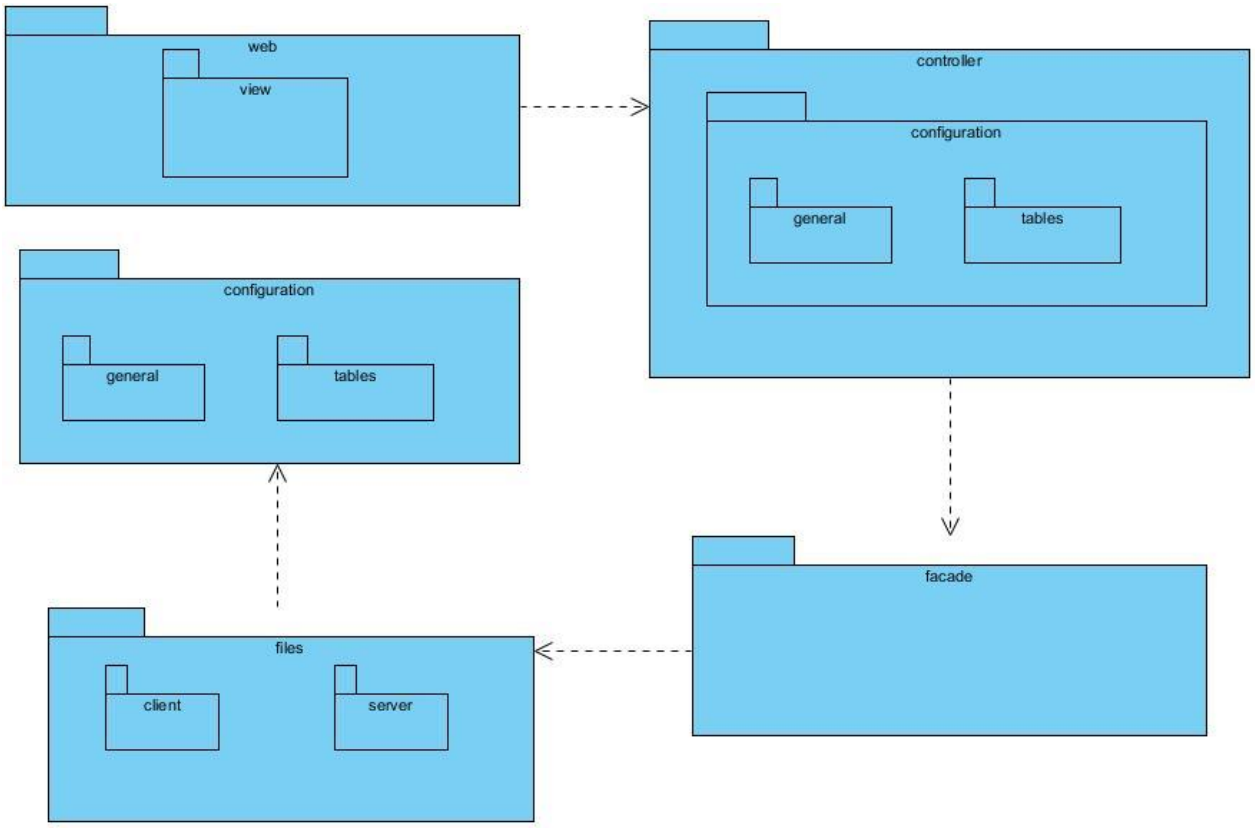


Figura 12 Diagrama de paquetes

2.8.2 Diagrama de clases del diseño

Un diagrama de clases de diseño muestra la especificación para cada una de las clases y las relaciones entre estas, utilizadas para la implementación del software. Incluye la siguiente información:

- Clases, asociaciones y atributos.
- Interfaces, con sus operaciones y constantes.
- Métodos.
- Navegabilidad.
- Dependencias.

A diferencia del modelo conceptual, un diagrama de clases de diseño muestra definiciones de entidades de software más que conceptos del mundo real. En la siguientes figuras se muestra los diagramas de clases del diseño correspondiente al servidor y al cliente [36].

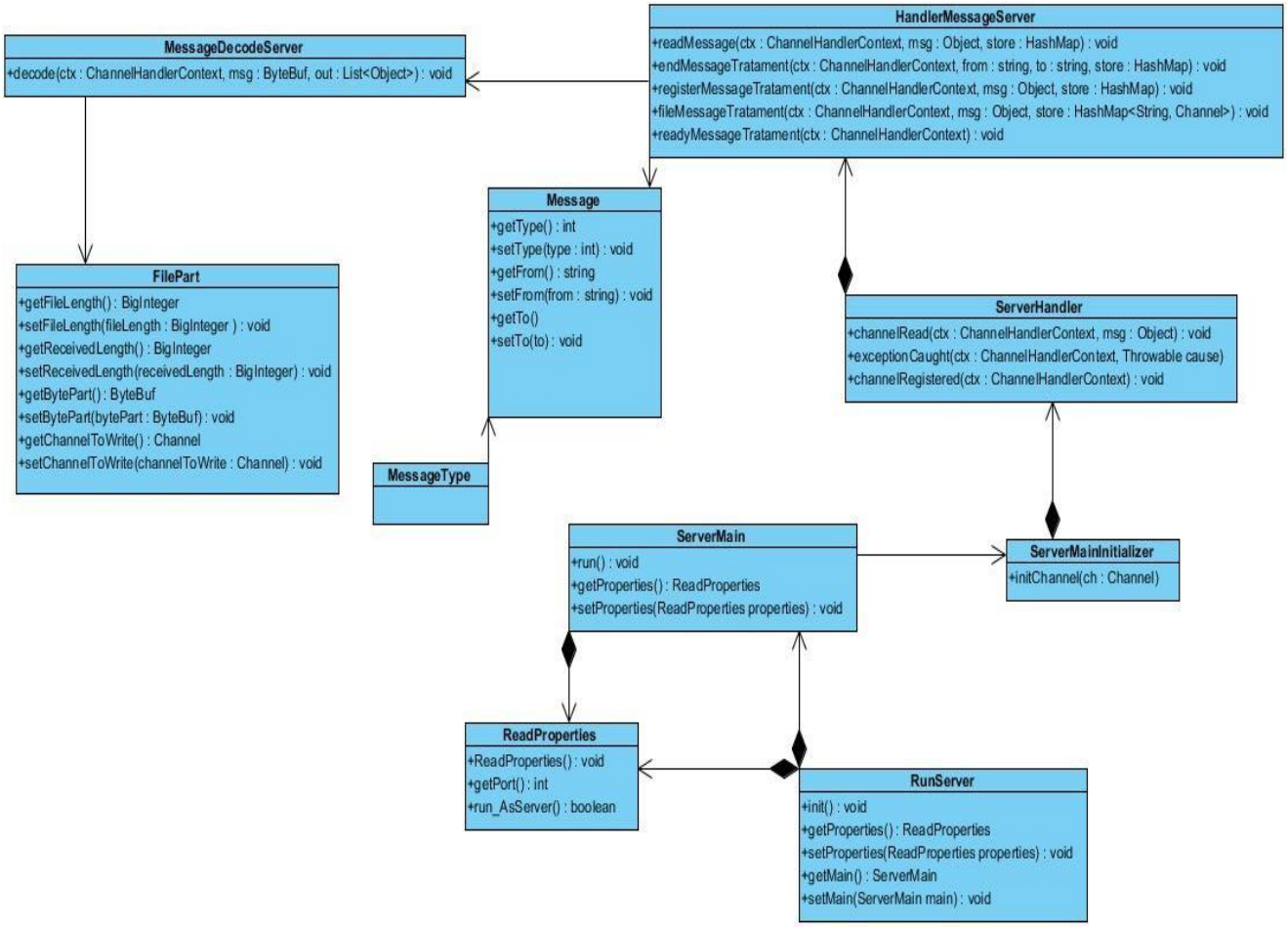


Figura 13 Diagrama de clases correspondiente al servidor

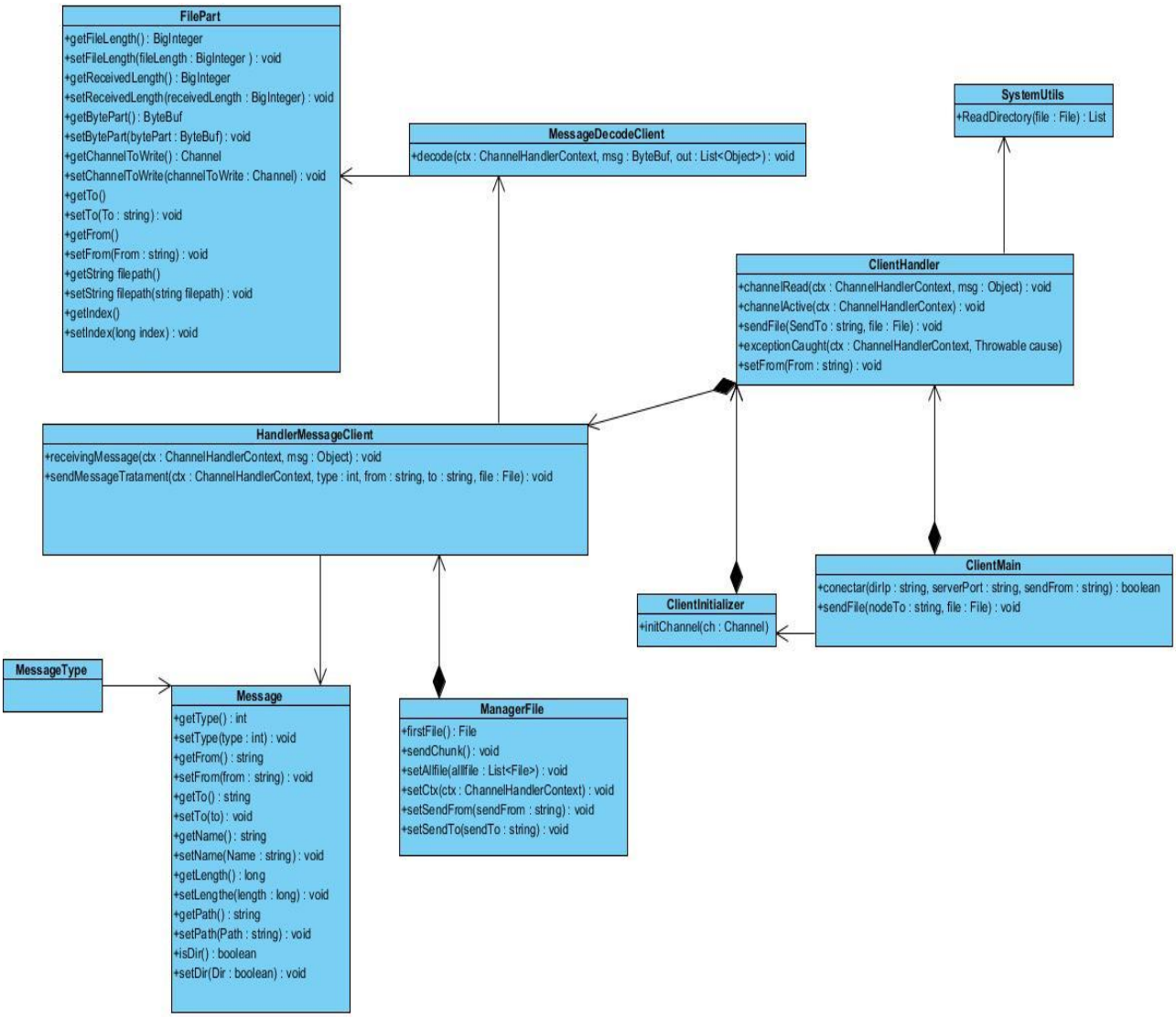


Figura 14 Diagrama de clases correspondiente al cliente

2.8.3 Descripción de las clases

A continuación, se realiza una breve descripción de algunas de las clases de la solución.

Tabla 10 Descripción de la clase ClientHandler

Descripción de la clase ClientHandler	
Nombre: ClientHandler	
Tipo de clase: Manejadora	
Atributos	**Tipos**
manejo	private

context	private
Responsabilidades	
Nombre:	channelRead(ChannelHandlerContext ctx, Object msg):void
Descripción:	Controla el flujo de información que recibe el cliente.
Nombre:	channelActive(ChannelHandlerContext ctx)
Descripción:	Maneja la actividad del cliente cuando tiene el canal activo.
Nombre:	sendFile (String sendTo,File file)
Descripción	Maneja el envío de los mensajes de información de los archivos.

Tabla 11 Descripción de la clase ClientInitializer

Descripción de la clase ClientInitializer	
Nombre: ClientInitializer	
Tipo de clase: Manejadora	
Atributos	**Tipos**
Responsabilidades	
Nombre:	initChannel(Channel ch):void
Descripción:	Encargada de crear el canal, con sus encoder y decoder y el handler que usara dicho canal.

Tabla 12 Descripción de la clase ManejarMessageClient

Descripción de la clase ManejarMessageClient	
Nombre: ManejarMessageClient	
Tipo de clase: Modelo	
Atributos	**Tipos**
from	private
to	private
filecreated	private
filetosave	private
ctx	private
Responsabilidades	
Nombre:	receivingMessage (ChannelHandlerContext ctx, Object msg):void
Descripción:	Maneja los mensajes recibidos por el cliente dependiendo del tipo que sea dicho mensaje.
Nombre:	sendMessage (ChannelHandlerContext ctx, int type, String from, String to, File file:void
Descripción:	Maneja el envío de mensajes desde el cliente.

Tabla 13 Descripción de la clase MessageDecoderServer

Descripción de la clase MessageDecoderServer	
Nombre: MessageDecoderServer	
Tipo de clase: Modelo	
Atributos	**Tipos**
length	private
receivedlength	private

channelToWrite	private
log	private
Responsabilidades	
Nombre:	decode (ChannelHandlerContext ctx, ByteBuf msg, List<Object> out): void
Descripción:	Recibe los arreglos de bytes y los guarda en un objeto de tipo FilePart que es la clase encargada de manejar los arreglos de bytes y crear los archivos con esos bytes.

Tabla 14 Descripción de la clase Message

Descripción de la clase Message	
Nombre: Message	
Tipo de clase: Modelo	
Atributos	**Tipos**
from	private
to	private
type	private
length	private
path	private
dir	private
name	private
Responsabilidades	
Nombre:	getName()
Descripción:	Devuelve el nombre del archivo a enviar.
Nombre:	setName(String name)

Descripción:	Establece el nombre del archivo.
Nombre:	getFrom()
Descripción:	Devuelve el nodo origen
Nombre:	setFrom(String From)
Descripción:	Establece el nodo origen
Nombre:	getTo()
Descripción:	Devuelve el nodo destino
Nombre:	setTo(String To)
Descripción:	Establece el nodo destino.
Nombre:	getType()
Descripción:	Devuelve el tipo de mensaje
Nombre:	setType(int type)
Descripción:	Establece el tipo de mensaje
Nombre:	getLength()
Descripción:	Devuelve el tamaño del fichero a enviar
Nombre:	setLength(long length)
Descripción:	Establece el tamaño del archivo a enviar

2.9 Modelo de despliegue

Un diagrama de despliegue modela la arquitectura en tiempo de ejecución de un sistema. Esto muestra la configuración de los elementos de hardware (nodos) y muestra cómo los elementos y artefactos del software se trazan en esos nodos[37].

En la figura 14 se muestra el diagrama de despliegue del software

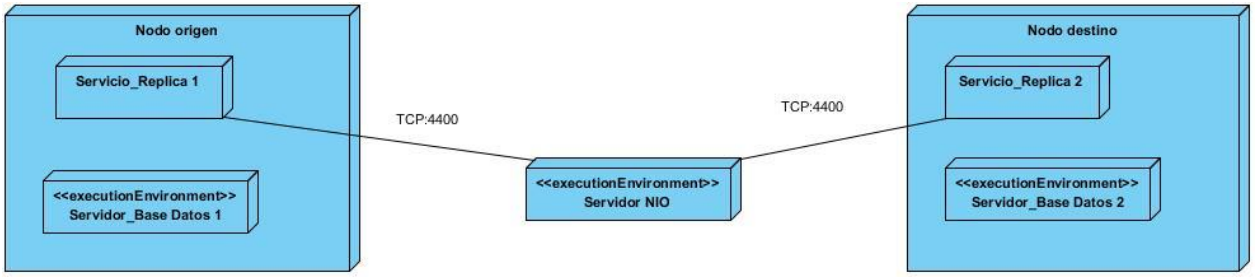


Figura 15 Modelo de despliegue

Nodo origen: es desde donde el archivo de gran tamaño es replicado y enviado hacia el servidor para que este lo re-direccione hacia el nodo destino que es donde tiene que llegar la réplica.

Nodo destino: es a donde llega la réplica del archivo, recibiendo los datos desde el servidor que es el intermediario de este envío.

Servidor de ficheros: Encargado de recibir los datos y redireccionarlos desde el nodo origen hacia el destino.

El nodo origen y destino pueden cambiar dependiendo de la función que se tenga que realizar, el destino puede llegar a ser origen y origen el destino, esta es la representación genérica del proceso de replicación de datos desde un nodo origen hacia un destino.

2.10 Consideraciones parciales del capítulo

- Con la correcta implementación de los requisitos extraídos se logrará conseguir un sistema capaz de darle solución al problema planteado.
- Con el modelado realizado se hace más fácil la tarea de implementación ya que se tiene claro toda la estructura que tendrá el proyecto y como estará compuesto.

Capítulo 3. Implementación y pruebas

Introducción:

A continuación, se muestran los estándares de codificación empleados en la solución, la implementación de los patrones expuestos en el capítulo anterior y se realizan pruebas al sistema para demostrar la validez de la implementación.

3.1 Estándares de codificación

Los estándares de código, son parte de las llamadas buenas practicas o mejores prácticas, estas son un conjunto no formal de reglas, que han ido surgiendo en las distintas comunidades de desarrolladores con el paso del tiempo y las cuales, bien aplicadas pueden incrementar la calidad del código, notablemente[38].

Entendemos como estándar de código a un conjunto de convenciones establecidas de ante mano (denominaciones, formatos, etc.) para la escritura de código. Estos estándares varían dependiendo del lenguaje de programación elegido y además varían en cobertura, algunos son más extensos que otros. Pero hay varios puntos que los estándares deberían cubrir[38].

A continuación, se mencionan algunos estándares empleados en la investigación

Estilos de comentarios

El estilo de los comentarios debe ser como el estilo de comentarios para Java (/* */ o //).

```
36
37      //Creando un objeto de la clase ManejarMessageCLient
38          ManejarMessageClient manejo= new ManejarMessageClient();
39          @Override
```

Figura 16 Estilo de comentarios

Identación:

La identación usada es la definida por el IDE IntelliJ IDEA 2016.3.3

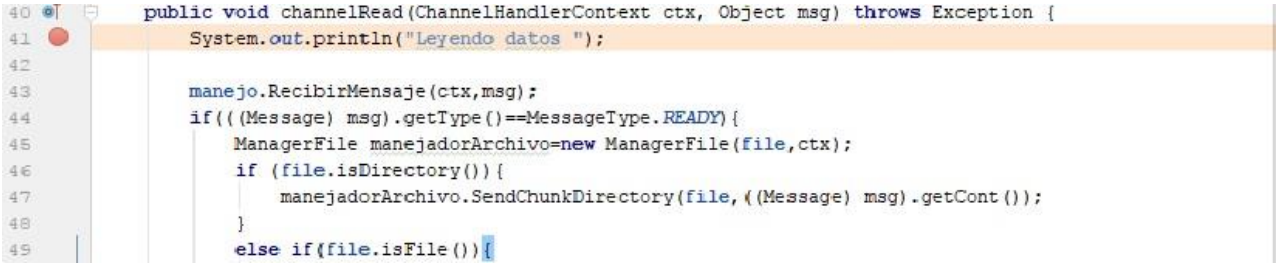


Figura 17 Estilo de identación

Declaración de variables

Para la declaración de variables se usó CamelCase, CamelCase es un tipo de escritura definida por el uso de mayúsculas y minúsculas, que se caracteriza porque las palabras van unidas entre sí sin espacios; con la peculiaridad de que las primeras letras de cada término se encuentran en mayúscula para hacer más legible el conjunto. Tradicionalmente, se había utilizado para la formulación química; pasando a ser empleado, en la actualidad, como un lenguaje de catalogación o clasificación, utilizado exclusivamente en la Web como lenguaje de programación o simplemente para llamar la atención con fines publicitarios.

El CamelCase admite dos posibles combinaciones entre mayúsculas y minúsculas:

- UpperCamelCase:la primera palabra en mayúscula y el resto en minúscula. LowerCamelCase: cuando la primera está en minúscula y las demás están en mayúscula. [39].

El autor del trabajo de diploma selecciona LowerCamelCase como estándar de codificación debido a que esta es la que se usa en el desarrollo del software replicador de datos REKO.

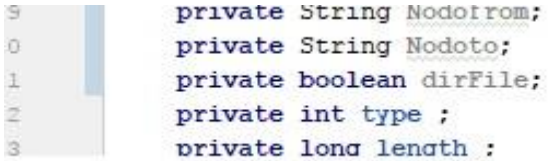


Figura 18 Estilo declaración de variables

Declaración de métodos

Siguen el mismo convenio que las variables.

```
76      public void sendMessage(ChannelHandlerContext ctx, int type, String from, String to, File file){
77          System.out.println("Activo");
78          Message message = new Message();
79          message.setFrom(from);
80          switch (type){
81              case MessageType.REGISTER:{
82                  message.setType(MessageType.REGISTER);
83              }break;
84              case MessageType.FILE:{
85                  message.setType(MessageType.FILE);
86                  message.setName(file.getName());
87                  message.setLength(file.length());
88                  message.setFrom(from);
```

Figura 19 Declaración de métodos

3.2 Implementación de los patrones de diseño

Patrón creador

Se usó en la clase ClientHandler la que tiene como responsabilidad crear un objeto de tipo ManejarMessageClient (clase) la cual se encarga sus métodos del trabajo con los mensajes que recibe el cliente.

```
32      public class ClientHandler extends ChannelInboundHandlerAdapter {
33          int cont =0;
34
35          File file = new File("D:\\reposiciongoal_the_dream_begins.webm");
36
37          //Creando un objeto de la clase ManejarMessageCLient
38          ManejarMessageClient manejo= new ManejarMessageClient();
39          @Override
40          public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
41              System.out.println("Leyendo datos ");
42
43              manejo.RecibirMensaje(ctx,msg);
44              if(((Message) msg).getType()==MessageType.READY){
45                  ManagerFile manejadorArchivo=new ManagerFile(file,ctx);
46                  if (file.isDirectory()){
47                      manejadorArchivo.SendChunkDirectory(file,((Message) msg).getCont());
48                  }
```

Figura 20 Uso del patrón creador

Patrón bajo acoplamiento y patrón alta cohesión

En este diagrama se ve que hay pocas dependencias entre las clases involucradas en el proyecto dando muestra del bajo acoplamiento y la relación entre ellas para resolver distintas tareas.

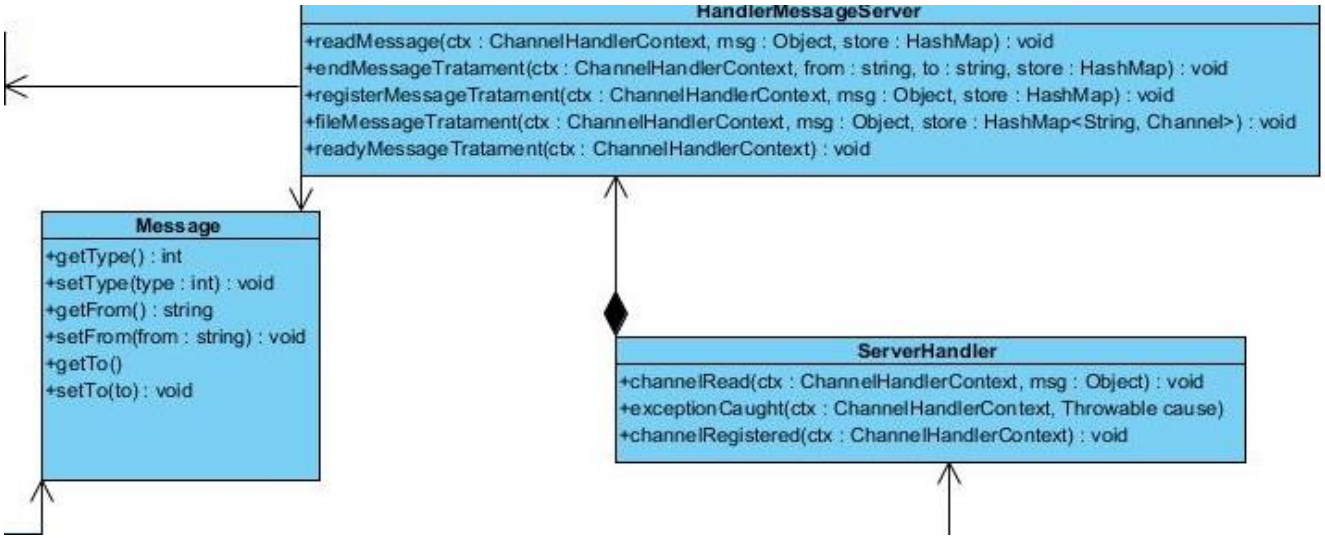


Figura 21 Uso del patrón bajo acoplamiento y del alta cohesión

3.3 Código fuente

A continuación, se muestra un fragmento de código perteneciente a la clase ***HandlerMessageServer***, el código fuente corresponde al método ***fileMessageTratament***, que se encarga de crear los mensajes con la información de los ficheros para enviarlo a los nodos destino y el método ***readMessage*** que es el que se encarga de manejar todos los mensajes entrantes.

Tabla 15 Código fuente de los métodos readMessage y fileMessageTratament

HandlerMessageServer

```
HandlerMessageServer | readyMessageTratament()
23  public void readMessage(ChannelHandlerContext ctx, Object msg, HashMap store) throws IOException, InterruptedException {
24      if(msg instanceof Message) {
25          switch (((Message) msg).getType()) {
26              case MessageType.REGISTER: {
27                  registerMessageTratament(ctx,msg,store);
28              }break;
29              case MessageType.FILE: {
30                  fileMessageTratament(ctx,msg,store);
31              }break;
32              case MessageType.READY:{
33                  readyMessageTratament(ctx);
34              }
35              case MessageType.END:{
36                  endMessageTratament(ctx,((Message) msg).getFrom(),((Message) msg).getTo(),store);
37              }break;
38          }
39      }else if(msg instanceof FilePart){
40          FilePart filePart = ((FilePart)msg);
41          Channel channel = filePart.getChannelToWrite();
42          channel.writeAndFlush(filePart.getBytePart()).addListener(future -> {
43              if(!future.isSuccess()){
44                  //Aqui tratamiento para cuando hay error en el envio del fichero;
45              }
46          });
47          if(filePart.getFileLength().compareTo(filePart.getReceivedLength())==0){
48              ctx.pipeline().remove( s: "files");
49              ctx.pipeline().addFirst("decode",new ObjectDecoder(ClassResolvers.cacheDisabled( classLoader: null)));
50          }
51      }
52
53  }
```

HandlerMessageServer

```
84      public void fileMessageTratament(ChannelHandlerContext ctx, Object msg,HashMap<String,Channel> store) throws IOException, InterruptedException {
85          System.out.println("Envio de Archivo");
86          boolean isDir=((Message) msg).isDir();
87          Long length=((Message) msg).getLength();
88          String Path=((Message) msg).getPath();
89          String OS=((Message) msg).getOS();
90          int cont=((Message) msg).getCont();
91          String path=((Message) msg).getPath();
92          String Name=((Message) msg).getName();
93          String from = ((Message) msg).getFrom();
94          String to = ((Message) msg).getTo();
95          Channel channelTosend =  store.get(to);
96          Channel channelToGet =  store.get(from);
97
98          if (channelTosend != null && channelToGet !=null ) {
99              if (channelTosend.isActive() && channelTosend.isWritable()) {
100                 System.out.println("Pidiendo Archivo al " + from);
101                 Message message = new Message();
102                 message.setType(MessageType.FILE);
103                 message.setFrom(from);
104                 message.setName(Name);
105                 message.setPath(path);
106                 message.setTo(to);
107                 message.setLength(length);
108                 BigInteger length1 = new BigInteger(String.valueOf(((Message) msg).getLength()));
109                 channelTosend.writeAndFlush(message).addListener(new ChannelFutureListener() {
110                     @Override
111                     public void operationComplete(ChannelFuture channelFuture) throws Exception {
112                         if(channelFuture.isSuccess()) {
113                             Message messageReady = new Message();
114                             messageReady.setType(MessageType.READY);
115                             channelToGet.writeAndFlush(messageReady);
116                             channelToGet.pipeline().addFirst("files", new MessageDecodeServer(length1,channelTosend));
117                             channelToGet.pipeline().remove( s: "decode");
118                         }else{
119                             //Aqui poner tratamiento para en casi de error
120                         }
121                     }
122                 });
123             }
124         }else {
125             System.out.println("Nodo no se encuentra");
126             Channel channelToResponse = channelToGet;
127             Message response = new Message();
128             response.setType(MessageType.ERRORNODE);
129             ChannelFuture future = channelToResponse.writeAndFlush(response);
130
131         }
132     }
```

3.4 Fase de pruebas

Las pruebas de software son un elemento crítico para la garantía de la calidad del software y representan una revisión final de las especificaciones, del diseño y de la codificación. El objetivo fundamental de las pruebas es descubrir diferentes clases de errores con la menor cantidad de

tiempo y de esfuerzo. Aunque las pruebas no pueden asegurar la ausencia de defectos; sí pueden demostrar que existen defectos en el software[40].

Estas actividades se planean con anticipación y se realizan de manera sistemática. Cuando se aplican pruebas a un software es necesario tener en cuenta el objetivo que se persigue, debido a que las pruebas son agrupadas por niveles que se encuentran en distintas etapas del proceso de desarrollo.

3.4.1 Pruebas de aceptación

Las pruebas de aceptación son pruebas de caja negra que se crean a partir de las historias de usuario. Durante las iteraciones las historias de usuario seleccionadas serán traducidas a pruebas de aceptación. En ellas se especifican, desde la perspectiva del cliente, los escenarios para probar que una HU ha sido implementada correctamente. Una HU puede tener todas las pruebas de aceptación que necesite para asegurar su correcto funcionamiento. El objetivo final de estas es garantizar que los requerimientos han sido cumplidos y que el componente es aceptable.

Las pruebas de aceptación tienen más peso que las unitarias ya que constituyen un indicador de la satisfacción del cliente con la solución.

Como resultado de las pruebas de aceptación se obtendrán artefactos descritos en tablas, estas contarán con los siguientes campos:

- **Código:** servirá como identificador de la prueba realizada, a su vez será sugerente al nombre de la prueba a la que hace referencia.
- **UH**: tendrá el nombre de la historia de usuario a la que hace referencia la prueba a realizar.
- **Nombre**: nombre que se le da a la prueba a realizar.
- **Descripción**: se describe la funcionalidad que se desea probar.
- **Condiciones de Ejecución**: mostrará las condiciones que deben cumplirse para poder llevar a cabo el caso de prueba, estas condiciones deben ser satisfechas antes de la ejecución del caso de prueba para que se puedan obtener los resultados esperados.
- **Entradas / Pasos de Ejecución**: se hará la descripción de cada uno de los pasos seguidos durante el desarrollo de la prueba, se tendrá en cuenta cada una de las entradas que hace el usuario con el objetivo de ver si se obtiene el resultado esperado.
- **Resultado esperado**: se hará una breve descripción del resultado que se espera obtener con la prueba realizada.
- **Evaluación de la prueba**: acorde al resultado de la prueba realizada se emitirá una evaluación sobre la misma. Esta evaluación tendrá uno de los tres resultados que a continuación se describen:

- Bien: cuando el resultado de la prueba es exactamente el esperado por el usuario.
- Parcialmente bien: cuando el resultado no es completamente el esperado por el cliente o usuario de la aplicación y muestra resultados erróneos o fuera de contexto.
- Mal: cuando el resultado de la prueba realizada genera un error de codificación en la aplicación o muestra como resultado elementos no deseados o fuera de contexto, trayendo como consecuencia que la funcionalidad requerida por el cliente no tenga resultado, lo que invalida también la UH.

A continuación, se muestran algunos de los casos de pruebas de aceptación realizados para las historias de usuario que fueron establecidas en el capítulo anterior, las demás se encuentran en el Anexo 2

Tabla 16 Prueba de aceptación envío y recepción de ficheros modificados P1

Caso de Prueba de Aceptación	
Código: P1_ HU4,5,6	**Historia de Usuario:** 4,5,6
Nombre: Envío y recepción de ficheros modificados.	
Descripción: Prueba para el proceso de envío y recepción de un fichero modificado.	
Condiciones de Ejecución: Deben haberse realizado previamente una configuración de la conexión del servidor de ficheros y una configuración de réplica .	
Entradas/Pasos de Ejecución: Se modifica el nombre de un fichero en el directorio que se está escuchando, previamente definido en la configuración de la réplica.	
Resultado Esperado: El sistema debe identificar el cambio, enviarlo desde el nodo origen y recibirlo en el nodo destino, para así concluir el proceso de réplica.	
Evaluación de la Prueba: Bien.	

Tabla 17 Prueba de aceptación envío y recepción de ficheros modificados P2

Caso de Prueba de Aceptación	
Código: P2_ HU4,5,6	**Historia de Usuario:** 4,5,6
Nombre: Envío y recepción de ficheros modificados.	
Descripción: Prueba para el proceso de envío y recepción de un fichero modificado.	
Condiciones de Ejecución: Deben haberse realizado previamente una configuración de la conexión del servidor de ficheros y una configuración de réplica .	

Entradas/Pasos de Ejecución: Se cambia la información dentro del fichero en el directorio que se está escuchando, previamente definido en la configuración de la réplica.
Resultado Esperado: El sistema debe identificar el cambio, enviarlo desde el nodo origen y recibirlo en el nodo destino, para así concluir el proceso de réplica.
Evaluación de la Prueba: Bien.

Tabla 18 Prueba de aceptación envío y recepción de ficheros modificados P3

Caso de Prueba de Aceptación	
Código: P3_ HU4,5,6	**Historia de Usuario:** 4,5,6
Nombre: Envío y recepción de ficheros modificados.	
Descripción: Prueba para el proceso de envío y recepción de un fichero modificado.	
Condiciones de Ejecución: Deben haberse realizado previamente una configuración de la conexión del servidor de ficheros y una configuración de réplica .	
Entradas/Pasos de Ejecución: Se adiciona un nuevo fichero en el directorio que se está escuchando, previamente definido en la configuración de la réplica.	
Resultado Esperado: El sistema debe identificar el cambio, enviarlo desde el nodo origen y recibirlo en el nodo destino, para así concluir el proceso de réplica.	
Evaluación de la Prueba: Bien.	

3.4.2 Pruebas funcionales

Con el objetivo de probar que el proceso de réplica de ficheros se cumple como se tiene previsto, se realizaron varias configuraciones de réplicas con distintos tipos de ficheros y tamaños. En cada replica la cantidad de nodos era diferente; se transmitió la información desde nodos destinos hasta nodos origen.

Tabla 19 Archivos enviados 1 Iteración

Archivos replicados	Tamaño del Archivo	Extensión	Réplica exitosa	Fallo réplica
3	<1 MB	.doc	3	0

2	>1MB	.doc	1	1
2	>1MB	.pdf	1	1
5	>1MB	Otras extensiones	1	4

Finalizada la primera iteración de pruebas se detecta un error de complejidad media en el proceso de la reconstrucción del fichero en el nodo origen, debido a que el decodificador de los arreglos de bytes perdía algunos de estos envíos y el archivo llegaba corrupto.

El error detectado fue solucionado, ejecutándose satisfactoriamente el proceso de réplica en la segunda iteración de pruebas, llegando a replicar archivos mayores de 1 GB.

Tabla 20 Archivos enviados 2 Iteración

Archivos replicados	Tamaño del Archivo	Extensión	Réplica exitosa	Fallo réplica
3	<1 MB	.doc	3	0
2	>1MB	.doc	2	0
2	>1MB	.pdf	2	0
5	>1MB	Otras extensiones	5	0

Con el objetivo de evidenciar las mejoras funcionales de la presente solución con respecto a la anterior se hizo una comparación teniendo en cuenta los siguientes parámetros o aspectos: cantidad de archivos enviados, tamaño del archivo y el tiempo de envió de estos en un proceso de réplica entre la nueva solución usando el framework Netty y el existente usando el servidor de mensajería ActiveMQ:

Tabla 21 Comparación de réplicas

	Archivos enviados	Tiempo de envío		Archivos enviados	Tiempo de envío
Netty			ActiveMQ		
<1MB	20	50 seg	<1MB	20	4 min
>1 MB	20	2 min	>1MB	20	10 min
>1GB	20	7 min	>1GB	20	30 min

3.5 Resultados de las pruebas funcionales

Luego de haber realizado las pruebas funcionales en cada una de las iteraciones con sus correspondientes casos de prueba, se obtuvieron los siguientes resultados:

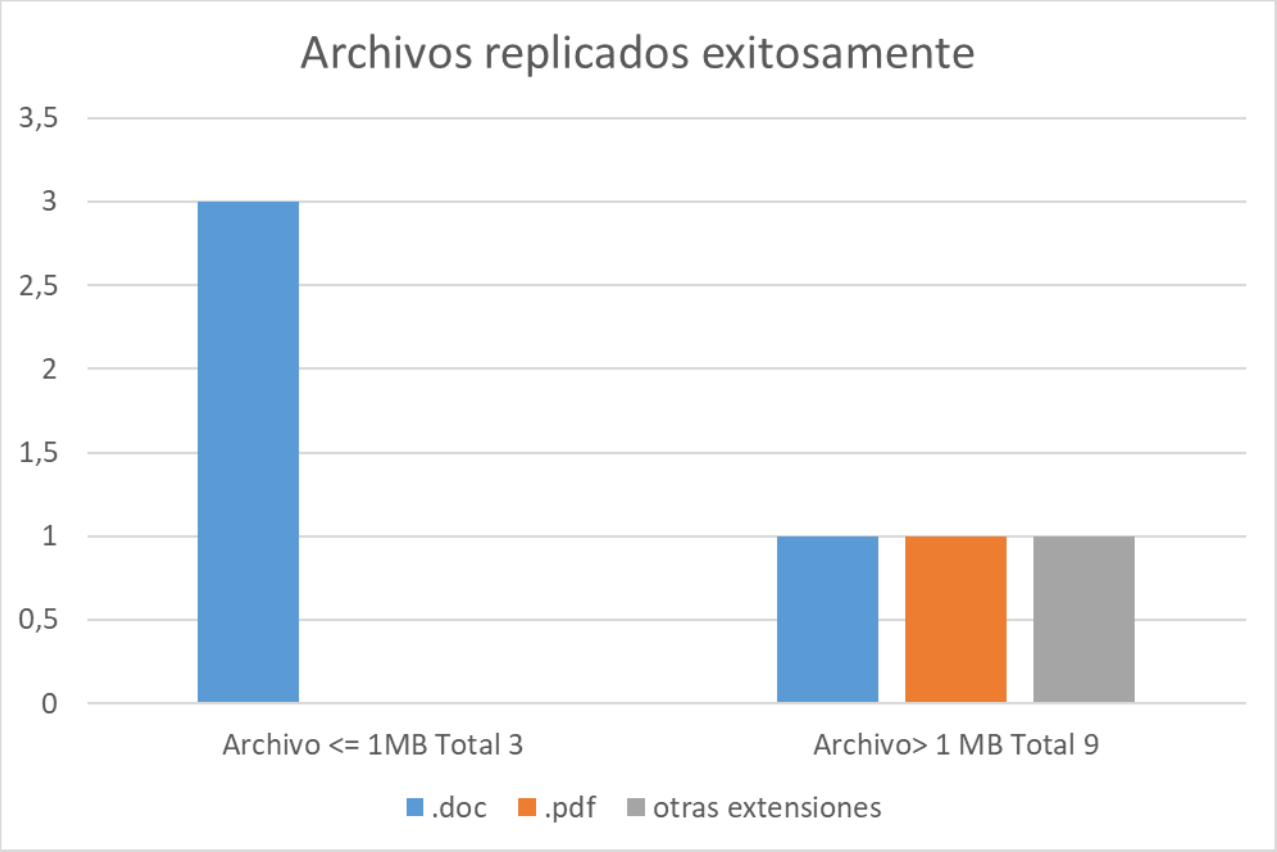


Figura 22 Resultados de las pruebas

En la figura 21 se puede verificar que en una primera iteración los archivos de menos de 1MB los 3 ficheros enviados se replicaron exitosamente y los mayores de 1MB de un total de 9 se replicaron exitosamente 1 .doc, 1 .pdf y 1 de otras extensiones, ocurriendo fallos en 6 de estas.

3.6 Resumen de las pruebas

Con la realización de las pruebas se pudo demostrar que la solución planteada es una mejora notable con respecto a la anterior, así como que la solución cumple con todas las funcionalidades que se definieron en los requerimientos.

3.7 Consideraciones parciales del capítulo

- Las funcionalidades implementadas garantizan el proceso de envío de ficheros de gran tamaño a través de nodos usando el Servidor NIO como puente entre ellos.
- Las realizaciones de las pruebas validaron las funcionalidades de la solución implementada estando acordes a los requerimientos planteados.
- Los resultados arrojados por las pruebas reflejan la robustez de la solución desarrollada.

CONCLUSIONES GENERALES

- La elaboración del marco teórico fue la fuente para recopilar información y sintetizar los conocimientos necesarios que garantizaron dar inicio a la investigación y proponer una correcta solución.
- La solución desarrollada garantiza el envío de ficheros en el proceso de réplica de ficheros, con lo cual minimiza la generación de errores que detienen el proceso de réplica.
- Las pruebas realizadas a la solución validaron su correcto funcionamiento y su aceptación por parte del cliente.
- Con esta solución se separa el proceso de réplica de ficheros del replicador REKO, de la réplica de datos como tal, ya que se deja de hacer uso del ActiveMQ como medio de comunicación.

RECOMENDACIONES

Se recomienda extender el mecanismo de comunicación de la réplica de fichero de tal manera que se pueda utilizar también en la réplica de datos.

BIBLIOGRAFÍA REFERENCIADA

1. Chavez, O. *Administracion de Bases de Datos*. [cited 2016 6 de diciembre]; Available from: http://chavez-atienzo-2013.blogspot.com/2013/04/replicacion.html

2. Colmenarez, B.L.y.R. *Bases de datos distribuidas* [cited 2016 22 de enero]; Available from: https://modelosbd2012t1.wordpress.com/2012/03/08/bases-de-datos-distribuidas/.

3. Vigo, D.d.I.d.S.y.A.U.d., *Ficheros*.

4. Comer, D.E., *Redes Globales de información con Internet y TCP/IP*.

5. Berzal, F., *Comunicacion entre procesos*.

6. Anon. *Protocolos de comunicaciones industriales*. [cited 2016 octubre]; Available from: <http://www.aie.cl/files/file/comites/ca/articulos/agosto-06.pdf>.

7. GARCÍA DE JALÓN, J.a.A., Iker and MORA, Alberto. *Aprenda LINUX como si estuviera en primero*. [cited 2016 octubre].

8. Garcia, F.M., *Unix.Programación Avanzada*. 3 ed.

9. Roldán, D.P., *Sistema de Clonación y Administración Centralizada de Imágenes de Sistemas Operativos*, Universidad de las Ciencias Informáticas: Ciudad de la Habana.

10. ALVAREZ, M.A., MONTEIRO, Juliana, JUSZKIEWICZ, Leo, WONG, William and Alvarez,Sara. *Tutorial de FTP*. Available from: <http://www.wiener.edu.pe/manuales2/5to-ciclo/REDES-2/manual-tutorial-ftp.pdf>.

11. Marshall, D. *Remote Procedure Calls (RPC)*. [cited 2016 octubre]; Available from: <http://www.cs.cf.ac.uk/Dave/C/node33.html>.

12. Winer, D. *What is XML-RPC?* [cited 2016 octubre]; Available from: <http://www.xmlrpc.com/>.

13. Maurer, N., *Netty in Action*. 5 ed. 2013.

14. *Netty*. Available from: https://netty.io/index.html.

15. Foundation, T.A.S. *Apache Mina Project*. Available from: https://mina.apache.org/.

16. Muguercia, E.B., *Sincronización de datos del proceso de réplica de estructura para el Replicador de datos REKO*, in *CDAE*. 2015, Universidad de las Ciencias Informaticas.

17. Autores, C.d., *Metodología de desarrollo para la Actividad productiva de la UCI.* 2014.

18. *What is Java ?* ; Available from: http://searchsoa.techtarget.com/definition/Java.

19. Russ Miles, K.H., *Learning UML 2.0*, I. "O'Reilly Media, 2006, Editor. 2006.

20. Walls, C., *Spring in Action*. Vol. 3. 2011: Manning Publications. 426.

21. Foundation, J. *What is IntelliJ IDEA Community Edition?* ; Available from: http://www.jetbrains.org/pages/viewpage.action?pageId=983211.

22. Gracia, L.M. *Netty: Librería de comunicación con Sockets NIO*. [cited 2016 Diciembre]; Available from: https://unpocodejava.wordpress.com/2010/06/25/netty-libreria-de-comunicacion-con-sockets-nio/.

23. Larman, P.H.C., *MODELO DEL DOMINIO*.

24. Pressman, R.S., *Ingeniería del Software Un enfoque práctico*. Vol. VI.

25. M.Fuentes, *MATERIAL DIDÁCTICO NOTAS DEL CURSO ANÁLISIS DE REQUERIMIENTOS.*

26. PMOinformatica.com, C.d.a. *Requerimientos no funcionales: Ejemplos* Available from: http://www.pmoinformatica.com/2015/05/requerimientos-no-funcionales-ejemplos.html.

27. Suaza, K.V., *Definición de equivalencias entre historias de usuario y especificaciones en UNLENCEP para el desarrollo ágil de software.* 2013.

28. Quintero, J.B., *Arquitectura de Software:Definiciones y Contexto*.

29. Mestras, J.P., *Patrones de diseño orientado a objetos*, Dep. Ingeniería del Software e Inteligencia Artificial Universidad Complutense Madrid.

30. Montero, R.U. *¿Qué es un Patrón de Diseño?* ; Available from: http://osc.co.cr/%C2%BFque-es-un-patron-de-disen/.

31. Lopez, E. *Arquitectura de n capas*. Available from: http://www.academia.edu/10102692/Arquitectura_de_n_capas.

32. Alem, F. *Patrones de Diseño (GRASP)*. 2015; Available from: http://www.taringa.net/posts/apuntes-y-monografias/18339620/Patrones-de-Diseno-GRASP.html.

33. Angelfire. *Patrones GoF*. Available from: http://geektheplanet.net/5462/patrones-gof.xhtml.

34. ARELI, T.H., *FUNDAMENTOS DE INGENIERIA DE SOFTWARE*.

35. Gutierrez, D., *UML Diagrama de Paquetes*. 2009.

36. *Diagrama de Clases de Diseño*. Available from: http://stadium.unad.edu.co/ovas/10596_9836/diagrama_de_clases_de_diseo.html.

37. *Diagrama de Despliegue UML 2*. Available from: http://www.sparxsystems.com.ar/resources/tutorial/uml2_deploymentdiagram.html.

38. *Estándares de codificación*. Available from: https://www.ohmyroot.com/buenas-practicas-legibilidad-del-codigo/.

39. LucioMSP. *{CamelCase}*. Available from: https://vicenteguzman.mx/2016/09/28/camelcase/.

40. Gonzalez, C.M.F., *Componentes gráficos para la representación de válvulas y bombas en el SCADA SAINUX.*, UNiversidad de las Ciencias Informaticas.

BIBLIOGRAFÍA

1. *Spring Tool Suite 3.7.1 released.* **Lippert, Martin.** October 06, 2015.

2. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns.* ISBN 0-201-63361-2.

3. **Hitchens, Ron.** *Java NIO.* s.l. : Editorial O´Reilly. ISBN:0596002882.

4. **C.J.Date.** *Introduccion a los Sistemas de bases de datos.* ISBN:968-444-419-2.

5. **Graba, Jan.** *An Introduction to Network Programming with Java.* s.l. : Springer. ISBN:978-1-44715253-8.

6. **Reinoso, Carlos Billy.** *Introduccion a la Arquitectura de Software.*

7. *Java How to Program.* **H.M, Deitel.** s.l. : Prentice Hall, 2006, Vol. VII. ISBN:9780136085676.

8. **Sharan, Kishori.** *Beginning Java 8 Fundamentals.* ISBN:978-1-4302-6658-7.

9. **Marin, Robert C.** *Codigo Limpio Manual de estilo para el desarrollo ágil de software.*

10. **Pressman, Roger S.** *Ingeniería del Software Un enfoque práctico.* Vol. VI.

11. **Raoul-Gabriel Urma, Mario Fusco,Alan Mycroft.** *Java 8 IN ACTION.* ISBN:978-161-729-199-9.

12. **P.J.Deitel, H.M.Deitel.** *Java,How to Program.* Seventh. ISBN:9780136085676.

13. **Arnaud Cogoluegnes, Thierry Templier,Andy Piper.** *Spring Dinamic Modules in Action.* ISBN:9781935182306.

14. **S.Pressman, Roger.** *Ingeniería del Software Un enfoque práctico.* Five.

15. **E.Comer, Douglas.** *Redes globales de informacióncon internet y TCP/IP.* ISBN:968-880-541-6.

16. ***Socket IO Real-time Web Aplication Development.*** **ISBN:978-1-78216-0786.**

17. Ron Morrison, Flavio Oquendo. ***Software Architecture.*** **ISBN:978-3-540-26275-6.**

18. Martin W.Murhammer, Orcun Atakan,Stefan Bretz,Larry R.Pugh. ***TCP/IP Tutorial and Technical Overview.***

19. Eduardo Alcalde, Jesús García Tomás. ***Introduccion a la TeleInformatica.*** **ISBN:8-1-481-01-0-5.**

20. Oscar Déniz Suárez, Alexis Quesada Arencibia,Francisco J.Santana Pérez. ***Comunicación mediante Sockets.***

21. Márquez, José E.Briceño. ***Transmisión de datos.***

ANEXOS

Anexo 1 Historias de Usuario

Historia de Usuario	
Número: HU3	**Nombre del requisito:** Guardar configuración de réplica de ficheros
Programador: Rogelio González Martínez	**Iteración asignada:** 1
Prioridad: Media	**Tiempo estimado:** 2 semanas
Riesgo en desarrollo: Bajo	**Tiempo real:** 10horas
Descripción: El sistema debe una vez que el usuario configure una nueva "Configuración de réplica" asignarle esta configuración a un nodo destino que es hacia donde se va a dirigir el proceso de réplica y guardar dicha configuración, y mostrar un mensaje de "Guardada exitosamente".	
Observaciones:	
Prototipo de interfaz:	
	

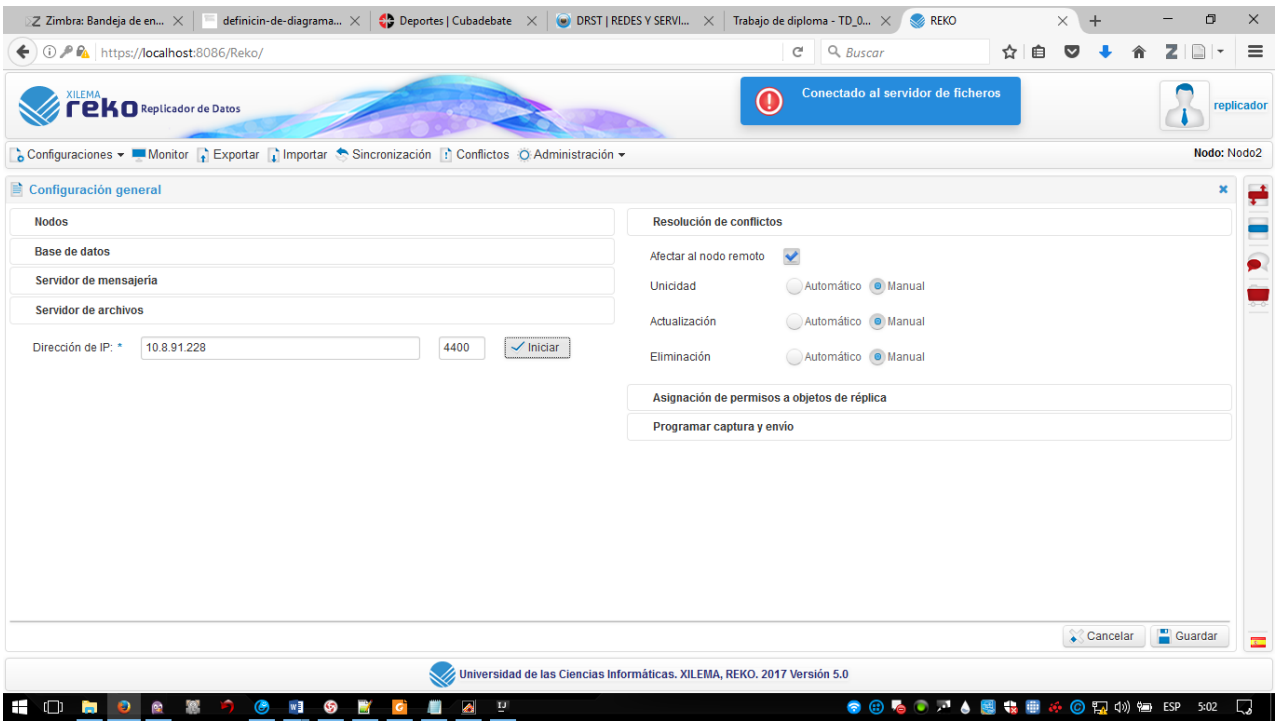
Historia de Usuario	
Número: HU4	**Nombre del requisito:** Capturar cambios en el fichero
Programador: Rogelio González Martínez	**Iteración asignada:** 1
Prioridad: Media	**Tiempo estimado:** 2 semanas
Riesgo en desarrollo: Bajo	**Tiempo real:** 10horas
Descripción: El sistema debe una vez que exista una configuración de réplica escuchando a un directorio, detectar los cambios ocurridos ya sea el cambio de nombre de un archivo, modificación de la información contenida en dicho fichero o que haya sido añadido algún nuevo fichero al directorio	

Observaciones:
Prototipo de interfaz:

Anexo 2: Pruebas de aceptación

Tabla 22 Prueba de conexión al servidor de ficheros P1

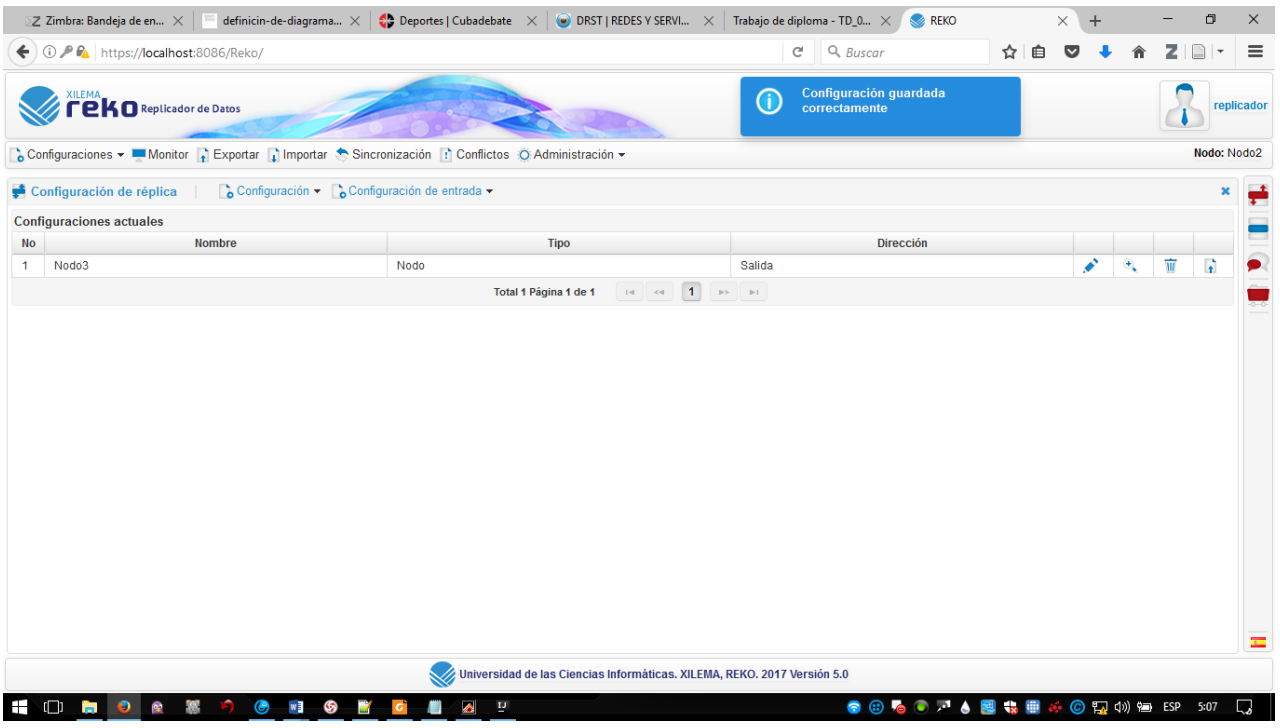
Caso de Prueba de Aceptación	
Código: P1_ HU1	**Historia de Usuario:** 1
Nombre: Conexión al servidor de ficheros	
Descripción: Prueba para la funcionalidad de conexión al servidor de ficheros	
Condiciones de Ejecución: Debe estar corriendo el sistema.	
Entradas/Pasos de Ejecución: El usuario en configuraciones generales , en la sección Servidor de archivos se debe introducir una dirección IP y un puerto en donde está corriendo el servidor encargado de la réplica de ficheros.	
Resultado Esperado: El sistema muestra un mensaje informando que se conectó correctamente al servidor de ficheros	



Evaluación de la Prueba: Bien.

Tabla 23 Prueba de configurar réplica de ficheros P1

Caso de Prueba de Aceptación	
Código: P1_ HU2	**Historia de Usuario:** 1
Nombre: Configurar réplica de ficheros	
Descripción: Prueba para la funcionalidad de configurar réplica de ficheros	
Condiciones de Ejecución: Debe estar corriendo el sistema.	
Entradas/Pasos de Ejecución: El usuario en configuración de réplica, en la sección Nueva se debe seleccionar el esquema en el cual se va a replicar y el modo de replica que en este caso será de fichero y seleccionado esto saldrá para definir el directorio al cual va estar escuchando el sistema para esa réplica, cuando todo esto está configurado se le da la opción de guardar y el sistema pide que defina cuál es el nodo destino de esta nueva replica .	
Resultado Esperado: El sistema configura la nueva réplica de ficheros satisfactoriamente y muestra un mensaje de esto, y sale la nueva configuración en la lista de configuraciones actuales.	



Evaluación de la Prueba: Bien.