Universidad de las Ciencias Informáticas
Facultad # 5

Título: “Arquitectura para el software de réplica de archivos RekoFile.”

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

Autor: Danieska Valladares Rodríguez.

Tutor: Ing. Albin Amat Reyes.

La Habana, 2012.
Año 54 de la Revolución.


"No hay más que asomarse a las puertas de la tecnología y la ciencia contemporánea para preguntarnos si es posible vivir y conocer ese mundo del futuro sin un enorme caudal de preparación y conocimientos"
Fidel Castro Ruz.

Dedicatoria

En especial quiero dedicar este trabajo a mi abuela Ermita Companioni que en paz descanse, por la que estoy eternamente agradecido por todo lo que hizo por mí, todo lo que soy hoy es por ella.

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

A mi papi Andrés, por darme las fuerzas para seguir adelante y el cariño para no extrañar tanto.

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

Agradecimientos

A mi mamá Elsida y a mi papá Andrés que me han apoyado en todo momento y han confiado en mí desde que era una niña, por enseñarme a saber por qué luchamos cada momento de nuestras vidas, por darme su amor y creer siempre en que yo si podía salir adelante. Por ustedes estoy aquí y se los agradeceré por siempre. Los Adoro.

A mi novio Yainier por la dedicación, la comprensión y la calma que me ha tenido y en especial por su amor. Por toda su ayuda en estos años mostrándome que cada momento vale.

A mi hermano por cubrirme en casa durante 5 largos años y llenar con amor mi ausencia.

A mi familia en general que sé que me quieren mucho y nunca han dudado de mí.

A mis suegros Ceida y José así como sus familias, que me apoyaron y siempre se han preocupado mucho por mí.

A las motos por apoyarme siempre en los buenos y malos momentos.

A la Revolución por darme la oportunidad de estudiar en una universidad de excelencia, por poner a mi alcance todos los recursos para mi formación como profesional.

A todos los que de una manera u otra han brindado su apoyo y han sido parte de este logro.

AGRADECIMIENTOS

Al tribunal que con sus sugerencias, preguntas y consejos me han ayudado a ganar confianza en mí misma para poder hoy ser la ingeniera que soy.

En especial a todos los presentes en este día tan especial para mí.

Danieska Valladares Rodríguez

Declaración de auditoría

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

Danieska Valladares Rguez	Ing. Albin Amat Reyes
_____	_____
Firma del Autor	Firma del Tutor

DATOS DE CONTACTOS

Nombre: Danieska
Apellidos: Valladares Rodríguez
Correo: dvrodriguez@estudiantes.uci.cu

Nombre: Albin
Apellidos: Amat Reyes
Título universitario: Ing. en Ciencias Informáticas, Profesor Instructor
Correo: aamat@uci.cu

Resumen

El presente trabajo se realizó a raíz de la necesidad de lograr la réplica de archivos que no se encuentren referenciados a través de una Base de Datos. Para darle solución a esta situación se decidió definir la Arquitectura de Software del sistema.

Dicha arquitectura cuenta con los elementos necesarios para el desarrollo del sistema. Ya que facilita y agiliza la implementación de una aplicación para el proceso de toma de decisiones en cualquier organización. Precisamente el objetivo de este trabajo radica en diseñar la arquitectura para el software de réplica de archivos RekoFile.

Para el desarrollo de la investigación se realizó un estudio bibliográfico de los patrones, estilos de diseño, lenguajes de descripción de la arquitectura, herramientas, tecnologías, metodologías y métodos existentes. Además, se realizó la evaluación de la arquitectura propuesta a través del método Architecture Tradeoff Analisys Method (ATAM) y la técnica Basada en Escenarios obteniendo resultados satisfactorios.

PALABRAS CLAVES

ATAM, Base de Datos, estilos de diseño, metodología, patrones, software, técnica

Índice

Tablas y figuras:

Introducción

En los inicios de la informática[1], la programación se consideraba un arte y se desarrollaba como tal, debido a la dificultad que entrañaba para la mayoría de las personas, con el tiempo se han ido descubriendo y desarrollando formas y guías generales, que dan solución a los problemas más populares en el desarrollo de una aplicación.

La arquitectura de software[2] como disciplina dentro de la informática engloba estos procedimientos, pues a semejanza de los planos de un edificio o construcción, esta indica la estructura, funcionamiento e interacción entre las partes del software.

Con el surgimiento de la Universidad de las Ciencias Informáticas (UCI), cuya meta es la formación de ingenieros en materia de desarrollo de software, y la creación de servicios y productos, la arquitectura de software pasó a ser un elemento importante en esta.

Con la creación de los centros de desarrollo como estrategia para potenciar y perfeccionar el desarrollo de software, surge el Centro de Consultoría y Desarrollo de Arquitecturas Empresariales (CDAE), cuya misión es brindar servicios de consultoría en tecnologías informáticas y desarrollar soluciones para organizaciones que buscan optimizar sus procesos de negocio y elevar la eficiencia operacional, empleando los modelos de Arquitectura Empresarial y SOA/BPM como paradigmas tecnológicos de referencia.

Una de las herramientas más exitosas desarrolladas en dicho centro es el software de réplica de Base de Datos Reko; que cubre las necesidades fundamentales de réplica de datos hacia diferentes localizaciones en los gestores de Bases de Datos relacionales más populares, sin embargo, debido a su creciente uso en diversos proyectos de aplicaciones distribuidas dentro de la red de centros de la UCI, sus funcionalidades iniciales han sido insuficientes para satisfacer las necesidades de réplica de muchos de estos proyectos.

[1] Disciplina que estudia el tratamiento automático de la información utilizando dispositivos electrónicos y sistemas computacionales. La unión sinérgica del cómputo y las comunicaciones.

[2] Conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica.

Una de estas necesidades constituye la réplica de archivos que no se encuentren referenciados por ninguna vía en la Base de Datos relacional. Debido a esto resulta imposible para dicho software, replicar estos archivos por lo cual se concibió la idea de la creación de RekoFile, un software integrado a Reko, capaz de replicar archivos y sistemas de archivos hacia otras localizaciones.

RekoFile al igual que Reko deberá permitirle al usuario cubrir las necesidades fundamentales de replicación de archivos tales como sincronización, transferencia de archivos entre diversas localizaciones y la centralización de estos en una única localización, la protección y la recuperación, así como cubrir otras necesidades relacionadas con la distribución de archivos. Además contará con la característica de ser multiplataforma por lo que el manejo de los sistemas de archivos dentro de los diferentes sistemas operativos constituirá una parte importante dentro de su implementación.

Una aplicación con todas estas características deberá basarse en una arquitectura robusta y madura que sea capaz de incorporar todas las funcionalidades requeridas para el cumplimiento de los objetivos de RekoFile, además de garantizar seguridad, integridad, portabilidad y escalabilidad.

La AS forma la columna vertebral para construir un sistema de software, es en gran medida responsable de permitir o no ciertos atributos de calidad del sistema entre los que se destacan la confiabilidad y el rendimiento del software. Además es un modelo abstracto reutilizable que puede transferirse de un sistema a otro y que representa un medio de comunicación y discusión entre participantes del proyecto, permitiendo así la interacción e intercambio entre los desarrolladores con el objetivo final de establecer el intercambio de conocimientos y puntos de vista entre ellos. [1]

De modo que para lograr una arquitectura confiable y robusta debemos encararla desde el siguiente **problema científico:** ¿Cómo lograr una arquitectura robusta y madura para el software RekoFile que permita la réplica de ficheros y garantice la organización de los componentes fundamentales del software?

La investigación tiene como **objeto de estudio** la Arquitectura de Software (AS) para sistemas de réplica, cuyo **campo de acción** está enmarcado en la arquitectura de software para sistemas de réplica de archivos.

De esta manera la **Idea a defender** con el presente trabajo:

Si se define una arquitectura de software robusta y madura para el sistema de réplica de archivos RekoFile, se garantizará la réplica de archivos y se cumplirá con los requisitos de seguridad, integridad, portabilidad y escalabilidad del sistema.

Objetivo General: Diseñar una arquitectura robusta, madura que permita al software de réplica RekoFile garantizando los requerimientos de seguridad, integridad, portabilidad[3] y escalabilidad[4] del sistema.

La consecución de tal objetivo estará sustentada en los **objetivos específicos** siguientes:

- Definir los estilos de arquitectura a utilizar para el desarrollo del software RekoFile.
- Determinar las funcionalidades más importantes desde el punto de vista arquitectónico.
- Definir la arquitectura a través de las diferentes vistas arquitectónicas (Vista general de un módulo, la vista física y la vista por componentes).
- Definir las herramientas y plataformas de desarrollo a utilizar para la implementación del software RekoFile.

Para alcanzar el objetivo propuesto se realizarán las siguientes **tareas**:

- Estudiar los enfoques y propuestas arquitectónicas de la bibliografía consultada, estableciendo comparaciones entre ellos, para obtener la más adecuada para la aplicación.
- Identificar los estilos y patrones arquitectónicos a utilizar, fundamentando los mismos y sus aplicaciones de manera que permitan un mínimo acoplamiento y un máximo de cohesión entre los componentes del sistema.
- Definir herramientas y estrategias a utilizar para el desarrollo de la aplicación.
- Diseñar la propuesta arquitectónica del sistema verificando que la misma permita la implementación de las funcionalidades deseadas y cumpla con los atributos de calidad requeridos.

[3] Características que posee un software para ejecutarse en diferentes plataformas.
[4] En la informática, propiedad deseable de un sistema que indica su habilidad para crecer o para manejar el crecimiento contínuo de manera fluida.

Para realizar las tareas se emplearon los siguientes **métodos**:

Los métodos científicos utilizados en esta investigación son los teóricos y los empíricos. Dentro de los métodos teóricos se utilizan los de análisis-síntesis, el histórico-lógico, y la modelación. Además cómo métodos empíricos están la observación.

A continuación se especifica el porqué de la selección de los mismos.
Análisis y Síntesis: para el procesamiento de la información y arribar a las conclusiones de la investigación, así como para precisar las características del modelo arquitectónico propuesto.
Histórico – Lógico: para determinar las tendencias actuales de desarrollo de los modelos y enfoques arquitectónicos.
Modelación: para la creación de modelos con vistas a investigar la realidad, en este caso se dará a conocer una propuesta de arquitectura de software que es un modelo.
Observación: para la percepción selectiva de las restricciones y propiedades del sistema y sistémica, para el control de la evolución de la arquitectura inicial.

Aporte Práctico, Resultados esperados o Beneficios.

Se espera obtener una arquitectura para el sistema de réplica de archivos RekoFile, flexible, orientada a los requisitos del sistema. Qué cumpla con los parámetros de funcionalidad, seguridad, portabilidad, eficiencia y confiabilidad.

Estructuración del trabajo

Para lograr una mejor comprensión el presente trabajo consta de introducción, tres capítulos, conclusiones, recomendaciones y referencias bibliografías.

Capítulo 1: Fundamentación teórica.
Se aborda una breve descripción de qué es la arquitectura, por qué es necesario realizarla, así como los diferentes estilos y patrones de diseño arquitectónico existentes y más usados, además de la metodología a emplear para realizar la documentación de la arquitectura de un sistema.

Capítulo 2: Descripción de la arquitectura.

Se realiza una descripción del tipo de arquitectura que va a tener el software de réplica de archivos RekoFile y se realiza la representación de la arquitectura. Además se realiza una descripción de las herramientas, tecnologías que se proponen para el desarrollo del sistema.

Capítulo 3: Evaluación del diseño arquitectónico propuesto.

En este capítulo se presentan la calidad de la arquitectura, atributos, métodos de evaluación de arquitecturas de software, evaluación de la arquitectura del sistema.

Capítulo 1: Fundamentación Teórica.

1.1 Introducción.

En este capítulo se analizan los aspectos fundamentales a tener en cuenta para el desarrollo de la propuesta de arquitectura del software RekoFile, basadas en el objeto de estudio (AS) y el campo de acción. Además de realizar un análisis de algunas definiciones de arquitectura de software, estilos arquitectónicos y las arquitecturas más comunes para seleccionar la más adecuada.

1.2 Definición de arquitectura de software:

Según Philippe Kruchten:

“La arquitectura de software, tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad, y disponibilidad”.[2]

Len Bass, Paul Clements y Rick Kazman plantean:

“La arquitectura de software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos.” [3]

Para la investigación se coincide con la propuesta oficial del documento Std 1471-2000 del Institute of Electrical and Electronics Engineers (IEEE[5]), que expresa:

“La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución”. [4]
La arquitectura de software se ve influenciada por muchos factores, como la plataforma funcional del software (arquitectura de hardware, sistema operativo,

[5] Asociación dedicada a la estandarización.

sistemas de gestión de base de datos, protocolos para comunicaciones en red), los bloques de construcción reutilizables de que se dispone, un marco de trabajo para interfaces gráficas de usuario, consideraciones de implantación, sistemas heredados, y requisitos no funcionales (rendimiento y fiabilidad).

Por tanto la arquitectura:

1. Es una vista estructural de alto nivel.
2. Define estilos o combinación de estilos para una solución.
3. Se concentra en los requisitos no funcionales.
4. Esencial para el éxito o fracaso de un proyecto.

1.3 Importancia y necesidad de definir una arquitectura.

Si se desea construir un software de réplica de archivos se necesita tener una visión común de sistemas en desarrollo. Por lo tanto se necesita una arquitectura para:

- Comprender el sistema.
- Organizar el desarrollo.
- Fomentar la reutilización.
- Hacer evolucionar el sistema.

1.4 Estilos arquitectónicos [5]

Los estilos arquitectónicos de software son arquitecturas de software comunes, marcos de referencias arquitectónicas, formas comunes o clases de sistemas. Estos se definen como las 4C:

- Componentes
- Conectores
- Configuraciones
- Restricciones (Constraints)

Estos permiten sintetizar estructuras de soluciones que luego serán refinadas a través del diseño.

A la hora de definir un estilo arquitectónico es necesario tener en cuenta el tipo de aplicación ya que puede imponer restricciones que acotan la interacción de los componentes, además se tiene en cuenta el patrón de organización general.

Algunos de los principales estilos arquitectónicos que se usan en la actualidad están divididos por Clases de Estilos las que engloban una serie de estilos arquitectónicos específicos. [6]

- **Estilos de Flujo de Datos**
  - Tuberías y Filtros
- **Estilos Centrados en Datos**
  - Arquitectura de Pizarra o Repositorio
- **Estilos de Llamada y Retorno**
  - Modelo – Vista – Controlador (MVC)
  - Arquitectura en Capas
  - Arquitectura Orientada a Objetos
  - Arquitectura Basada en Componentes
- **Estilo de Código Móvil**
  - Arquitectura de Máquinas Virtuales
- **Estilos Peer – To – Peer (punto a punto)**
  - Arquitectura Basada en Eventos
  - Arquitectura Orientada a Servicios
  - Arquitectura Basada en Recursos.

1.4.1 Estilos de Llamada y Retorno.

Los sistemas basados en este estilo utilizan un programa principal que tiene el control del sistema y varios subprogramas que se comunican con éste mediante el uso de llamadas. Esta familia de estilos enfatiza el uso de la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala.

- ✓ **Modelo – Vista – Controlador (MVC).** [7]

Es un patrón de arquitectura de software que se utiliza principalmente cuando es necesario modular la interfaz[6] de usuario, las reglas de negocio y el control de eventos. Separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

[6] Conjunto de métodos para lograr interactividad entre un usuario y una computadora.

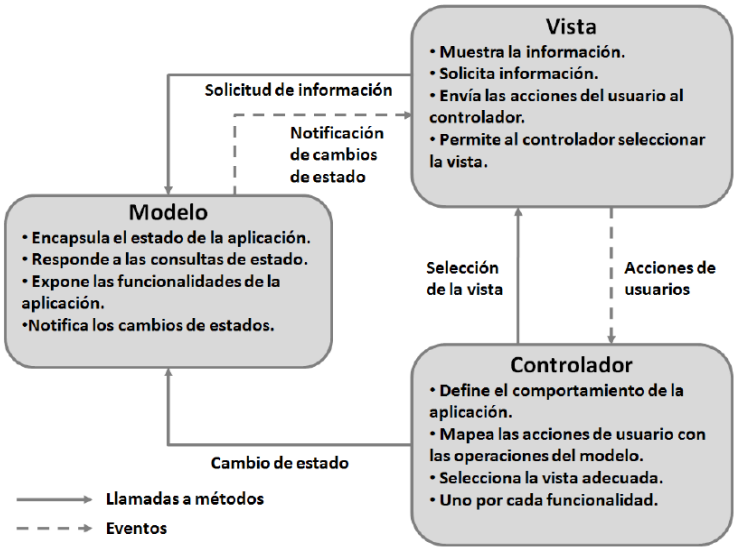

Figura 1: Modelo – Vista – Controlador

El modelo representa los datos y las reglas de negocio que rigen su acceso y actualización; puede verse como una modelación de los procesos del mundo real.

La vista se encarga de presentar los datos obtenidos del modelo. Es responsabilidad de las vistas mantener la información actualizada, esto se puede logar a través de peticiones de actualización al modelo o a través de notificaciones de cambio que el modelo emite (eventos).

Los controladores manipulan las entradas del usuario. La vista y el controlador conjuntamente comprenden la interfaz de usuario. Un mecanismo de propagación de cambio asegura consistencia entre la interfaz de usuario y el modelo.

Ventajas:

- Se pueden mostrar distintas variantes de interfaz gráfica simultáneamente.
- La interfaz tiende a cambiar más rápido que las reglas del negocio. Agregar nuevos tipos de vista no afecta el modelo.
- Evita poner código indebido en la capa impropia. Facilita despliegue en caso de modificaciones en el modelo de datos.

Desventajas:

- Puede aumentar un poco la complejidad de la solución. Como está guiado por eventos puede ser algo más difícil de depurar.
- La separación de conceptos en capas agrega complejidad al sistema.

- La cantidad de archivos a mantener y desarrollar se incrementa considerablemente.

✓ **Arquitectura en Capas.** [7]

La arquitectura en capa define el patrón en capas como una organización jerárquica. Lo que posibilita un diseño basado en niveles de abstracción creciente, posibilitando a los implementadores, particionar un problema en una secuencia de pasos incrementales. Este estilo de desarrollo en varios niveles, facilita que en caso que ocurra algún cambio, sólo se tendrían que realizar las correcciones necesarias en el nivel requerido sin tener que revisar código de otros niveles.

Capa de presentación: En esta capa se diseña todo lo que constituye la interfaz gráfica y la interacción del usuario con el software. Se comunica únicamente con la capa de negocio. Representa el conjunto de componentes que genera la información que se representará en la interfaz de usuario del cliente.

Capa de lógica de negocio: En esta capa residen los objetos de negocio que interactúan con los objetos de dominio. Se denomina capa de negocio (e incluso de lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse para un correcto funcionamiento lógico de la aplicación.

Capa de servicio: Esta capa permite desacoplar la interfaz de usuario del resto de capas, permitiendo que las funcionalidades de nuestra aplicación sean accesibles por otras aplicaciones u servicios.

Capa de Acceso a Datos: Encapsula la lógica de acceso a datos. Aquí se encuentran componentes que hacen transparente el acceso a la base de datos. Este es el lugar idóneo para implementar los objetos de acceso a datos, permitiendo ingresar, obtener, actualizar y eliminar información del Sistema de Bases de Datos.

Los Objetos de Acceso a Datos (DAO) encapsulan la persistencia de los objetos de dominio, proveen la persistencia de los objetos transitorios y las actualizaciones de los objetos existentes en la base de datos. Las implementaciones de los DAO estarán disponibles para los objetos de negocio.

Ventajas:

- El estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.
- El estilo admite muy naturalmente optimizaciones y refinamientos.
- Proporciona amplia reutilización. Al igual que los tipos de datos abstractos[7], se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas.

Desventajas:

- No admiten un buen mapeo[8] en una estructura jerárquica. Incluso cuando un sistema se puede establecer lógicamente en capas, consideraciones de rendimiento pueden requerir acoplamientos específicos entre capas de alto y bajo nivel.
- A veces es también extremadamente difícil encontrar el nivel de abstracción correcto, por ejemplo, la comunidad de comunicación ha encontrado complejo mapear los protocolos existentes en el framework ISO[9], de modo que muchos protocolos agrupan diversas capas, ocasionando que en el mercado proliferen los drivers[10] o los servicios monolíticos.
- Los cambios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel, en especial si se utiliza una modalidad relajada; también se admite que la arquitectura en capas ayuda a controlar y encapsular aplicaciones complejas, pero complica no siempre razonablemente las aplicaciones simples.

- ✓ **Arquitectura basada en componentes** [8]

Consiste en que uno o varios componentes en la aplicación, utilizan otros componentes que ya están definidos, se relacionan entre sí, y así proporcionan los servicios que se necesitan para que el sistema funcione. En este estilo las interfaces son el elemento básico de conexión entre los componentes, se debe conocer las interfaces que puede ofrecer cada componente, así como las que necesita para su operatividad, porque estas son el centro de la atención en el diseño arquitectónico. Cuando se desarrollan sistemas en los cuales la arquitectura sea de este tipo se trata

[7] Estructuras de datos genéricas, pueden soportar cualquier tipo de datos, generalmente son formas de ordenación de datos.
[8] Recorrer, acceder.
[9] Proyecto de la Organización Internacional para la Estandarización (ISO, por sus siglas en inglés), para la normalización de la construcción de lexicones y diccionarios de maquina (MRD).
[10] Software que permite al sistema operativo interactuar con el hardware.

de introducir todos los requerimientos funcionales en los componentes que forman parte del sistema. La conexión y coordinación entre los componentes se define mediante la arquitectura y estos deben corresponder con las necesidades de la arquitectura y del sistema. Un componente en este tipo de arquitectura representa una parte del software que puede ser reemplazada, por lo que debe poseer una baja dependencia con los demás componentes del sistema.

Además los componentes deben ser actualizados cada vez que se cambien los requisitos del sistema. En un sistema de software se pueden utilizar componentes que ya se hayan desarrollado con anterioridad y que se hayan utilizado en otros proyectos, o se pueden diseñar nuevos componentes para el sistema, los componentes nuevos deben ser comprobados y documentados. Esta arquitectura proporciona grandes ventajas para la calidad del software, ya que los gastos en el desarrollo del sistema no son tan elevados. A continuación se representa esta arquitectura.

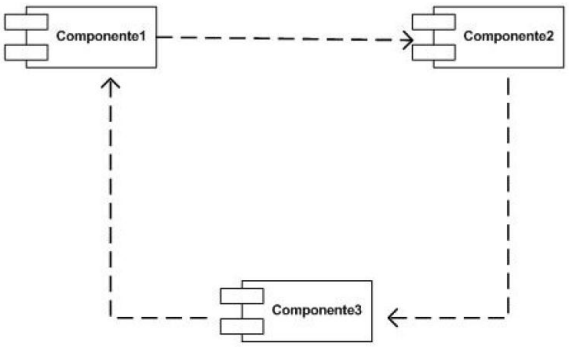

Figura 2: Representación de la Arquitectura basada en componentes.

- ✓ **Arquitectura Orientada a Objetos (OO):** [9]

Clase de componentes llamadas managers (gestoras), debido a que son responsables de preservar la integridad de su propia representación. Resumiendo las características de las arquitecturas OO[11], se podría decir que:

Los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo[12]. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación.

[11] Orientada a Objetos
[12] Capacidad que tienen los objetos de diferentes clases de responder al mismo mensaje en la programación OO.

Las interfaces están separadas de las implementaciones. En general la distribución de objetos es transparente, y en el estado de arte de la tecnología (lo mismo que para los componentes en el sentido de que apenas importa si los objetos son locales o remotos). El mejor ejemplo de OO para sistemas distribuidos es Common Object Request Broker Architecture, o Arquitectura Común de Intermediarios en Peticiones a Objetos (CORBA), en la cual las interfaces se definen mediante Interface Description Language, o Lenguaje de Especificación de Interfaces (IDL); un Object Request Broker media la interacción entre objetos clientes y objetos servidores en ambientes distribuidos.

Ventajas:

- Se puede modificar la implementación de un objeto sin afectar a sus clientes.
- Es posible descomponer problemas en colecciones de agentes en interacción.
- Un objeto es ante todo una entidad reutilizable en el entorno de desarrollo.

Desventajas:

- Para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad.
- Efectos colaterales en cascada: si A usa B y C también lo usa, el efecto de C sobre B puede afectar A.
- Entre las limitaciones, el principal problema del estilo se manifiesta en el hecho de que para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad. La consecuencia inmediata de esta característica es que cuando se modifica un objeto (por ejemplo, se cambia el nombre de un método, o el tipo de dato de algún argumento de invocación) se deben modificar también todos los objetos que lo invocan.

Luego de un análisis realizado a partir de los estilos anteriormente descritos y las características del sistema, de complejidad y alto nivel de diseño; se determinó que la aplicación requiere la separación en partes que puedan concentrarse en distintas áreas de funcionalidad. Por lo anteriormente mencionado se propone aplicar dentro de la familia de estilos de llamada y retorno, los estilos de arquitecturas basada en componentes y en capas.

La arquitectura basada en componentes lleva a alcanzar un mayor nivel de reutilización de software, permite que las pruebas sean ejecutadas probando cada uno

de los componentes antes de probar el conjunto completo de componentes ensamblados, simplifica el mantenimiento del sistema y tiene mayor calidad ya que un componente puede ser construido y luego mejorado continuamente por un experto u organización.

Esta arquitectura le permite a RekoFile poder aislar los componentes y mejorarlos cada uno por separado ya que utilizan una combinación del desarrollo interactivo e incremental y cada componente va a tener responsabilidades individuales pero conectadas entre sí; ya sea a través de objetos y mensajes formando un todo.

La arquitectura en capas mejora las posibilidades de mantenimiento, debido a que cada capa es independiente de la otra, los cambios o actualizaciones pueden ser realizados sin afectar la aplicación como un todo. Cada capa puede ser manejada y escalada de forma independiente, la flexibilidad se incrementa. Además al proporcionar una amplia reutilización de código se facilita la implementación de cada una de las capas de los subsistemas por parte de los desarrolladores.

1.5 Patrones

Un patrón de arquitectura de software es un esquema genérico probado para solucionar un problema particular, el cual es recurrente dentro de un cierto contexto. Este esquema se especifica describiendo los componentes, con sus responsabilidades y relaciones.

Los patrones arquitectónicos están relacionados con la interacción de los objetos dentro o entre niveles arquitectónicos, están dirigidos a la solución de problemas arquitectónicos como adaptabilidad a requerimientos cambiantes, funcionamiento, modularidad y acoplamiento, mientras las soluciones propuestas tienen que ver con patrones de llamadas entre objetos (similar a los patrones de diseño), decisiones y criterios arquitectónicos, empaquetado de funcionalidad, entre otros aspectos.

Los patrones pueden dividirse o clasificarse en diferentes tipos los utilizados dentro de la arquitectura del software de réplica de archivos RekoFile, son:

1.5.1 Patrones de Arquitectura [10]

Los patrones de arquitectura expresan el esquema fundamental de organización para

sistemas de software. Proveen un conjunto de subsistemas predefinidos; especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos. Los patrones de arquitectura representan el nivel más alto en el sistema de patrones. Ayudan a especificar la estructura fundamental de una aplicación. Cada actividad de desarrollo es gobernada por esta estructura; por ejemplo, el diseño detallado de los subsistemas, la comunicación y colaboración entre diferentes partes del sistema. Cada patrón de arquitectura ayuda a conseguir una propiedad específica en el sistema global.

- ✓ **Modelo Vista Controlador** [11]

Beneficios que brinda:

- **Menor acoplamiento.**

1. Desacopla las vistas de los modelos.
2. Desacopla los modelos de la forma en que se muestran e ingresan los datos.

- **Mayor cohesión.**

1. Cada elemento del patrón está altamente especializado en su tarea (la vista en mostrar datos al usuario, el controlador en las entradas y el modelo en su objetivo de negocio).

- **Las vistas proveen mayor flexibilidad y agilidad.**

1. Se pueden crear múltiples vistas de un modelo.
2. Se pueden crear, añadir, modificar y eliminar nuevas vistas dinámicamente.
3. Las vistas pueden anidarse.
4. Se puede cambiar el modo en que una vista responde al usuario sin cambiar su representación visual.
5. Se pueden sincronizar las vistas.
6. Las vistas pueden concentrarse en diferentes aspectos del modelo.

- **Más claridad de diseño.**
- **Facilita el mantenimiento.**
- **Mayor escalabilidad**.

- ✓ **Cliente- servidor:** [12]

La arquitectura cliente-servidor sustituye a la arquitectura monolítica en la que no hay distribución, tanto a nivel físico como a nivel lógico.

Ventajas de la arquitectura cliente-servidor:

- **Centralización del control:** Los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que un cliente web o no autorizado no pueda dañar el sistema.
- **Escalabilidad:** Indica su habilidad para crecer o para manejar el crecimiento continuo de manera fluida.

1.5.2 Patrones de Diseño

Los patrones de diseño son "la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces." Un patrón de diseño es una solución a un problema de diseño; facilitan la reusabilidad, extensibilidad y mantenimiento. [13]

✓ **Patrón de diseño DAO:** Plantea, utilizar un Data Access Object (DAO) para abstraer y encapsular todos los accesos a la fuente de datos, siendo el encargado de implementar el mecanismo de acceso requerido para trabajar con la misma. Como la interfaz expuesta por el DAO no depende de la fuente de datos subyacente, este patrón le permite adaptarse a diferentes esquemas de almacenamiento, sin que se vean afectados los componentes del negocio. Esencialmente, el DAO actúa como un adaptador entre el componente y la fuente de datos.

✓ **Patrones GRASP** [14]

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

Creador: Clases encargadas de instanciar a otras con responsabilidades. Dentro del manager de cada uno de los módulos se encuentran las funcionalidades definidas y se ejecutan. En cada una de estas funcionalidades se crean instancian las clases que representan las entidades, por lo que podremos que la clase interface manager es "creador" de esas entidades.

Controlador: Se encarga de asignar la responsabilidad de controlar el flujo de eventos del sistema a clases específicas, facilitando la centralización de actividades.

Experto: Encapsula toda la lógica de los datos y son generadas las clases con todas las funcionalidades comunes de las entidades.

Alta cohesión: Cada una de las clases representa una mínima dependencia con las demás clases al proponer un sistema estructurado en capas donde las inferiores no conocen la estructura de las de arriba y presentan una fachada con las funcionalidades precisas, para que las capas consumidoras no conozcan más que lo que deben, lo cual es garantizado con el estilo de 4 capas aplicado a RekoFile.

Patrones GOF [15]

Estructurales: Describen cómo las clases y los objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades.

- **Fachada (Facade):** Provee de una interfaz unificada simple, para acceder a una interfaz o grupo de interfaces de un subsistema. Es utilizado para reducir la dependencia entre clases, ofreciendo un punto de acceso, de manera que si estas cambian o se sustituyen por otras, solo hay que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones cliente. Fachada no oculta las clases, sino que ofrece una forma más sencilla de acceder a las mismas y en los casos que se requiera, permite acceder directamente a ellas.

1.6 Metodologías de Desarrollo de Software

El desarrollo de software puede ser un reto para los desarrolladores, pues crear un software en el menor tiempo posible y con calidad puede ser casi imposible, si no se cuenta con algún proceso que ayude a agilizar el desarrollo de software.

Desde hace algún tiempo se vienen utilizando las metodologías, ya que imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente.
Apoyando lo antes mencionado según Grady (2000), plantea que un proceso de desarrollo de software define quién está haciendo qué, cuándo y dónde para alcanzar un determinado objetivo; en la ingeniería el objetivo es construir un producto de software o mejorar uno existente con calidad. [16]

A continuación se tratan brevemente 3 de las metodologías más usadas en procesos de desarrollo de software en el mundo.

1.6.1 Proceso Unificado de Desarrollo (RUP)

El Proceso Unificado de Desarrollo (RUP por sus siglas del inglés Rational Unified Process), es una metodología para el desarrollo de software orientados a objetos. Es un proceso de desarrollo de software, definido como un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software.

RUP se puede considerar también un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organización y diferentes niveles de aptitudes.

Características específicas de RUP:

Dirigido por casos de uso: Esto significa que el proceso de desarrollo sigue una trayectoria que avanza a través de los flujos de trabajo generados por los casos de uso. Los casos de uso se especifican y diseñan al principio de cada iteración, y son la fuente a partir de la cual los ingenieros de prueba construyen sus casos de prueba. Estos describen la funcionalidad total del sistema.

Centrado en la arquitectura: Los casos de uso guían a la arquitectura del sistema y ésta influye en la selección de los casos de uso. La arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por las plataformas de software, sistemas operativos, sistemas de gestión de bases de datos, además de otros como sistemas heredados y requerimientos no funcionales. Se representa mediante varias vistas que se centran en aspectos concretos.

Iterativo e incremental: RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y las cuales se definen según el nivel de madurez que alcanzan los productos que se van obteniendo con cada actividad ejecutada. La terminación de cada fase ocurre en el hito correspondiente a cada una, donde se evalúa que se hayan cumplido los objetivos de la fase en cuestión. [17]

1.6.2 Programación Extrema (XP)

La Programación Extrema (XP por sus siglas en inglés Xtreme Programing), es una metodología ligera de desarrollo de software que se basa en la simplicidad, la

comunicación y la realimentación o reutilización del código desarrollado. La metodología consiste en una programación rápida o extrema, utilizadas para proyectos de corto plazo.

Características de XP, la metodología se basa en:

Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que se puede adelantar en algo hacia el futuro, se pueden hacer pruebas de las fallas que pudieran ocurrir. Es como si se obtuvieran los posibles errores.

Refabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa.

El desarrollo bajo XP tiene **características** que lo distinguen claramente de otras metodologías:

- Diseñadores y programadores se comunican efectivamente con el cliente y entre ellos mismos.
- Diseño de software sencillo y libre de complejidad o pretensiones excesivas.
- Retroalimentación de usuarios y clientes desde el primer día gracias a las baterías de pruebas.
- El software es liberado en entregas frecuentes tan pronto como sea posible.
- Los cambios se implementan rápidamente tal y como fueron sugeridos.
- Las metas en características, tiempos y costos son reajustadas, permanentemente en función del avance real obtenido.

1.6.3 Proceso unificado abierto (Open Up/Basic)

Dentro de las metodologías existentes se ha decidido utilizar Open Up que es un Framework de procesos de desarrollo de software de código abierto.

Es un proceso modelo y extensible, dirigido la a gestión y desarrollo de proyectos de software basados en desarrollo iterativo, ágil e incremental; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo.

Open UP/Basic se centra en articular la arquitectura para facilitar la colaboración técnica, reducir el riesgo y minimizar el sobreesfuerzo de desarrollo. Procura un equilibrio entre las necesidades de los involucrados con los resultados del proyecto y los costos técnicos, con el fin de maximizar el valor de los involucrados y las guías del proceso de desarrollo. [18]

Teniendo en cuenta lo antes planteado se realiza un ciclo de vida interactivo que mitiga el riesgo a tiempo y ofrece demostrar resultados en curso al cliente del proyecto.

Características Generales

- ✓ Preserva la esencia del Unified Process
  - Desarrollo iterativo e incremental
  - Desarrollo dirigido por Casos de Uso
  - Centrado en la Arquitectura

- ✓ Sólo lo fundamental está incluido, sin dejar de ser completo y extensible (menos de 20 artefactos) .

- ✓ Está pensado para proyectos pequeños.

Beneficios en el uso del Open Up

- Permite disminuir las probabilidades de fracaso.
- Incrementa las probabilidades de éxito.
- Permite detectar errores tempranos a través de un ciclo iterativo.
- Evita la elaboración de documentación, diagramas e iteraciones innecesarios requeridos en la metodología RUP.
- Por ser una metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas.

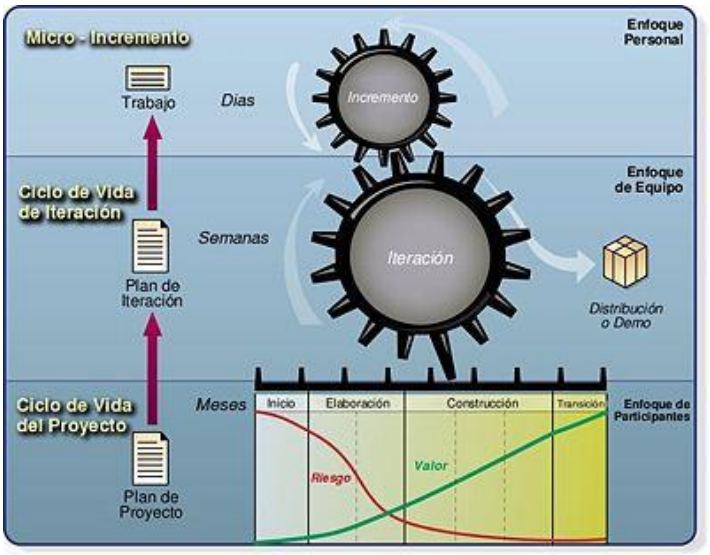

Figura 3: Ciclo de vida Open Up.

Fases de Open UP [19]

Concepción: Primera de las 4 fases que se centra en obtener toda la información relacionada con el proyecto y capturar las necesidades de los stakeholder (clientes) en los objetivos del ciclo de vida para el proyecto.

Elaboración: Definir los riesgos significativos para la arquitecturas, además de establecer la base para la elaboración de la arquitectura del sistema.

Construcción: Esta fase está enfocada al diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de esta fase es completar el desarrollo del sistema basado en la Arquitectura definida.

Transición: Es la última fase, cuyo propósito es asegurar que el sistema es entregado a los usuarios y evalúa las funcionalidades del último entregable de la fase de construcción.

1.7. Lenguaje de Modelado

UML

El Lenguaje Unificado de Modelado (UML por sus siglas en inglés) es una notación para especificar, visualizar y documentar sistemas de software desde la perspectiva

orientado a objetos. Su objetivo es representar el conocimiento acerca de los sistemas que se pretenden construir y las decisiones tomadas durante su desarrollo, tanto los representados por diagramas estáticos (Casos de Uso, diagrama de clases, etc.) como los dinámicos (Diagramas de actividades, interacción, etc.) La representación en UML de un sistema de software consta de cinco vistas o modelos parciales separados, aunque relacionados entre sí, denominadas vista de casos de uso, de lógica, de implementación, de procesos y de despliegue. Cada uno de estos modelos representa el sistema por medio de diversos diagramas.

Aunque no existe de forma explícita una vista arquitectónica, estas cinco vistas pretenden describir, en su conjunto, la arquitectura del sistema. [20]

Ventajas

- Es un lenguaje distribuido y adecuado a las necesidades de conectividad actual y futura.
- Está apoyado por la OMG (Object Management Group) como la notación estándar para el desarrollo de proyectos informáticos.
- Es útil para el desarrollo de modelaje visual de cualquier proyecto no solo informático y más aún es estándar.
- Modela estructuras complejas.
- Las estructuras más importantes que soportan tienen su fundamento en las tecnologías orientadas a objetos, tales como objetos, clase, componentes y nodos.
- Emplea operaciones abstractas como guía para variaciones futuras, añadiendo variables si es necesario. [20]

1.7 Conclusiones parciales

Este capítulo es el resultado de la búsqueda y el análisis de la información vinculada al objeto de estudio (AS) y los sistemas existentes asociadas al campo de acción. Después de haber realizado el estudio de cada uno de estos temas y teniendo en cuenta las características del componente fueron seleccionadas la arquitectura basada en componentes en conjunto con la arquitectura en capas. Como metodología de desarrollo se seleccionó Open UP, utilizando UML como lenguaje de modelado.

Capítulo 2: Descripción de la arquitectura.

2.1 Introducción

En este capítulo se analiza la estructura de la arquitectura del software de réplica de archivos RekoFile mediante la lógica de los componentes y su agrupación en distintas capas y niveles que se comunican entre sí y la propuesta de las herramientas a utilizar.

2.2 Vista General de la Arquitectura de RekoFile

El sistema está dividido en módulos, tal y como se muestran en la figura 4, en divisiones lógicas, agrupando funcionalidades afines a un módulo, está diseñado para agregar módulos automáticamente de forma que puedan integrarse dentro de la arquitectura sin necesidad de un mantenimiento a esta.

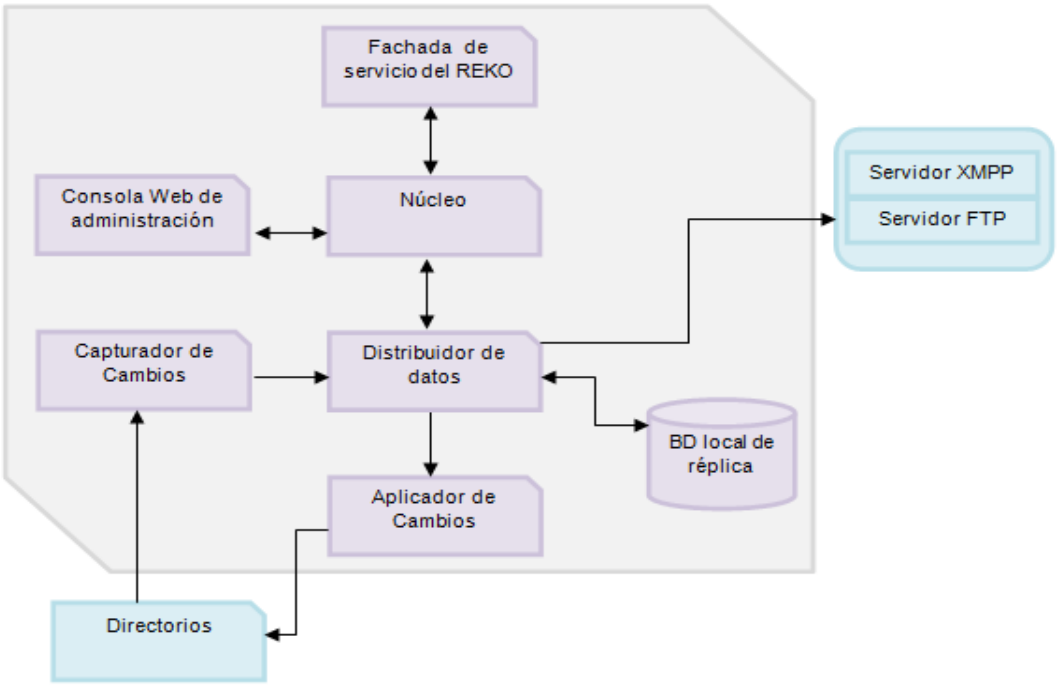

Figura 4: Vista General de la Arquitectura de RekoFile

Como se observa en la figura el sistema contará:

Fachada de servicio del REKO: Es una capa de comunicación con Reko para utilizar diversas funcionalidades.

Núcleo: Maneja las configuraciones y agrupa las principales funcionalidades.

Capturador de Cambios: Captura los cambios que se realizan sobre los ficheros y se los entrega al Distribuidor de Datos.

Aplicador de Cambios: Ejecuta sobre el fichero los cambios que sean replicados.

Distribuidor de Datos: Determina el destino de cada cambio realizado en los ficheros, los envía y se responsabiliza con su llegada.

BD Local de Réplica: Se utiliza para guardar las configuraciones propias de la réplica, las acciones sobre los ficheros que han dado conflicto al aplicarse y las acciones o transacciones que no han podido llegar a su destino.

Interfaz Web de Administración: Representa la interfaz del software. Permite realizar las configuraciones principales del software como el registro de nodos, ficheros a replicar y el monitoreo del funcionamiento del software.

Servidor XMPP: Protocolo de mensajería instantánea capaz de rastrear la presencia de un usuario en tiempo casi real. El protocolo también admite la comunicación de voz y transferencia de archivos.

Servidor FTP: Se utiliza de forma opcional en el funcionamiento del software. Su función principal es enviar grandes archivos de réplica y que se pueda resumir la transmisión de los mismos en caso de problemas en la conexión.

Directorios: Concepto que representa los directorios que se van a replicar. Los cambios ejecutados sobre ellos serán enviados, así como serán aplicados otros cambios provenientes de otros nodos de réplica.

2.3 Vista general de un módulo de RekoFile

Un módulo es un conjunto de funcionalidades agrupadas convenientemente para realizar un conjunto de procesos del negocio comunes dentro del sistema. Cada uno de los módulos de RekoFile deberá tener una estructura lógica en 4 capas: Presentación, Negocio, Servicio, Acceso a Datos.

La figura 5 muestra el flujo de datos a través de las capas en el módulo desde que el usuario

realiza una petición, la cual es capturada en la capa de presentación usando los controles JSP y elementos de tipo JavaScript de la librería Dojo.

Esta capa genera un grupo de eventos que son capturados dentro de la capa del negocio a través de las funcionalidades que ofrece la fachada del negocio, pasando luego las peticiones a la capa de gestión llamada manager la cual se encarga de procesar las peticiones de acuerdo con la lógica del módulo y decide si hacer o no peticiones a la capa de acceso a datos que es la encargada de interactuar con la base de datos del sistema.

Una vez procesada la solicitud, la respuesta se devuelve a la capa de presentación que la presenta de forma organizada, atractiva y entendible para el usuario. Para el caso de que el módulo forme parte de una funcionalidad para sistemas externos se establece una fachada de servicios que funciona de la misma manera que la capa de presentación dándole a otros sistemas la posibilidad de interactuar con el sistema y viceversa.

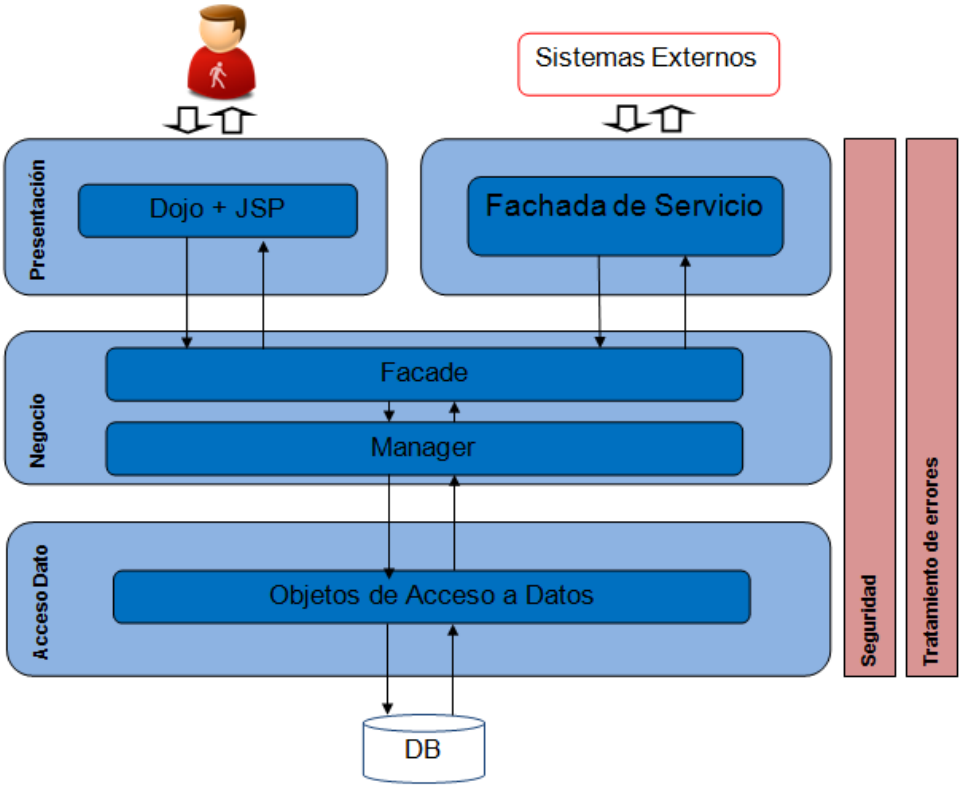

Figura 5: Vista general de un módulo de RekoFile

2.4 Vista física de RekoFile.

Una característica objetiva de los sistemas web es que son sistemas distribuidos, siguiendo el patrón arquitectónico cliente – servidor. Como se muestra en la figura 6 RekoFile puede

estar dividido en dos niveles físicos fundamentales: Clientes, Servidor, este último a su vez y en dependencia de la complejidad de dicho sistema, puede dividirse en Servidor Web y Servidor de base de datos Berkeley.

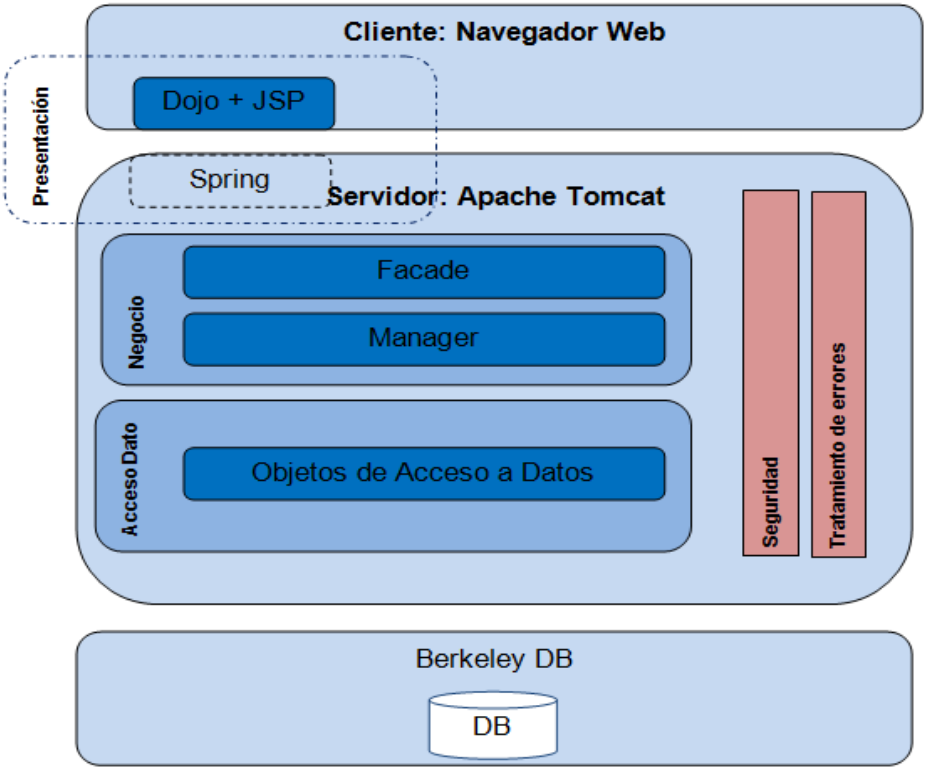

Figura 6: Vista física de RekoFile.

2.5 Vista por componentes:

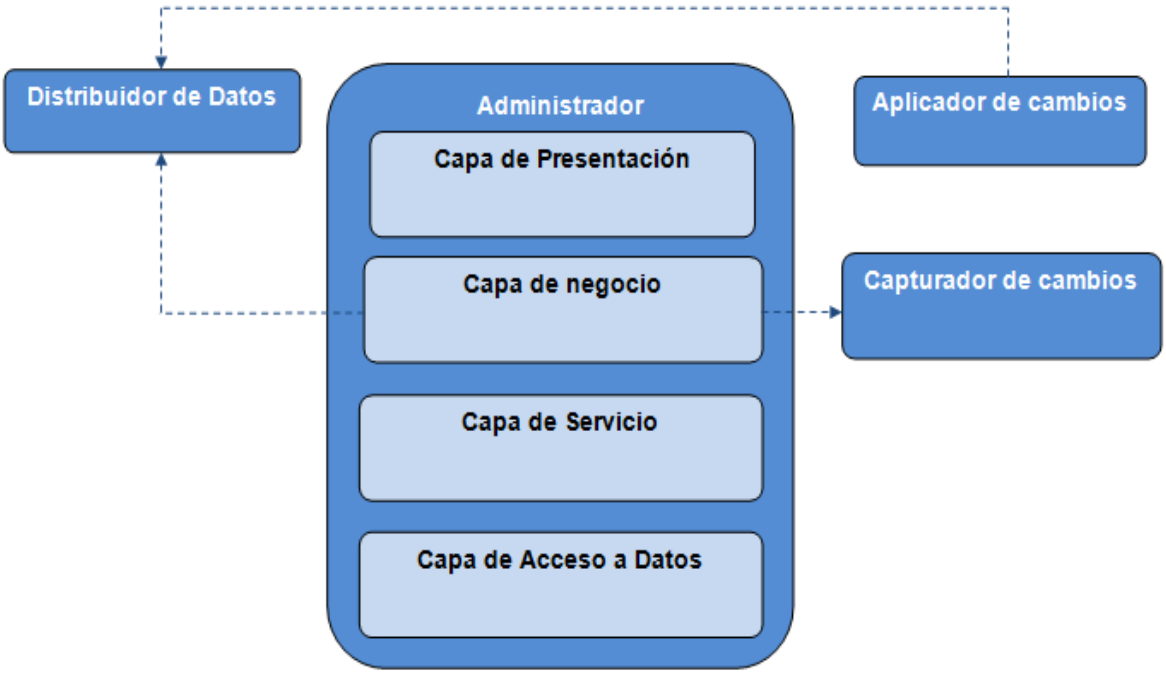

Figura 7: Vista por componentes

Los principales componentes presentes en el software son:

2.5.1 Capturador de cambios: Encargado de capturar los cambios que se realizan y entregarlos al distribuidor.

2.5.2 Distribuidor: Determina el destino de cada cambio realizado, los envía y se responsabiliza de su llegada.

2.5.3 Aplicador: Ejecuta en el fichero los cambios que son enviados hacia él desde otro nodo de réplica.

2.5.4 Administración: Permite realizar las configuraciones principales del software como el registro de nodos, configuración de los ficheros a replicar y el monitoreo del funcionamiento.

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

2.5.4.1 Capa de Presentación

La capa de presentación implementa y muestra los elementos de la interfaz de usuario (UI por sus siglas en inglés), controlando todo lo referente a la interacción con el usuario y las validaciones lógicas de los datos que se manejan.

Los componentes de UI proporcionan una forma para que los usuarios interactúen con la aplicación, mostrando los datos en el formato adecuado. Se encargan de obtener y validar los datos entrados por el usuario. Para la implementación de la presentación se utilizaron los componentes de UI de JSP y Dojo.

Para su entendimiento lógico se dividió la capa de presentación en dos partes, la parte en el cliente; vista en la parte superior de la figura 8 , donde se realiza la interacción con el usuario mediante controles de entrada y salida de datos y procesos de interfaz de usuario. El control de la interfaz de usuario es mayormente escrito en JavaScript, realizándose aquí también todas las funcionalidades relacionadas con la lógica de las validaciones. Además como medida de seguridad se realizará la validación de los datos del lado del servidor independientemente de la validación en el cliente.

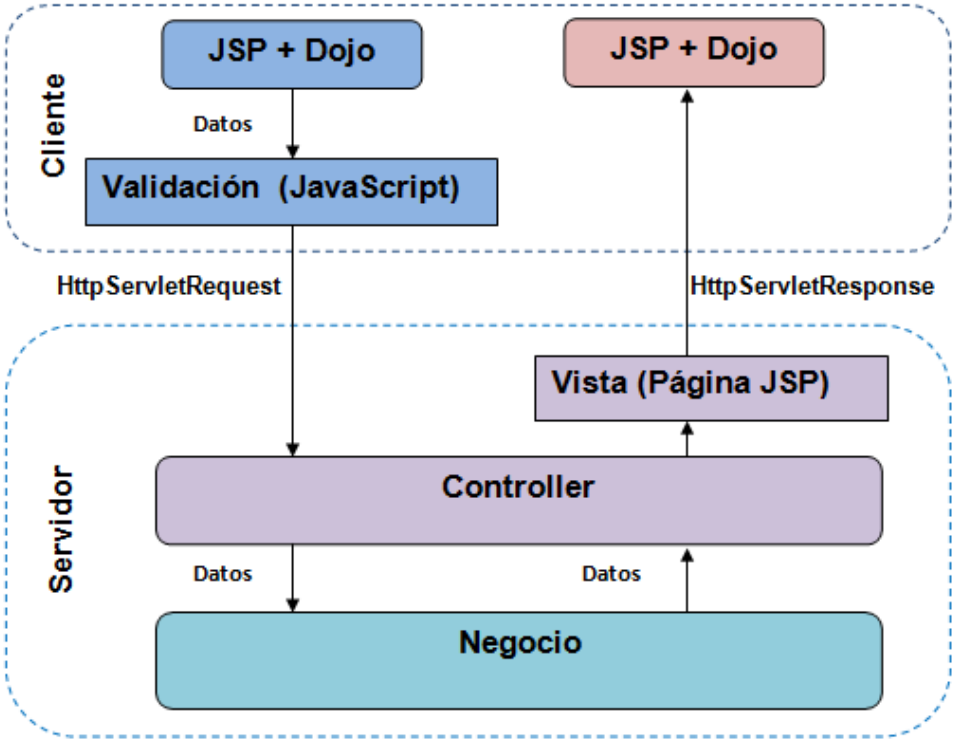

Figura 8: Capa de Presentación

2.5.4.2 Capa de Negocio

En esta capa y como se ve en la figura 9 se implementan las funcionalidades principales del sistema y se establecen las reglas del negocio más relevantes. Esta capa se compone de dos partes fundamentales: Las entidades del negocio y los componentes del negocio (Manager). Los componentes del negocio servirán al mismo tiempo como fachada del negocio en el que cada módulo controlará la forma en que se utilizan las funcionalidades en la capa de presentación y la de acceso a datos.

Las entidades del negocio se utilizan para trasmitir datos entre componentes. Estos datos se encapsulan en entidades informativas o contenedores que representan los objetos o fenómenos del mundo real y su valor fundamental residen en la información que contiene y no en su comportamiento en sí. Para cada entidad que exista en el módulo se crea una clase manejadora (manager) que es la encargada de implementar la lógica del negocio de dicha entidad. El manager se encargará también de interactuar con la capa de acceso a datos.

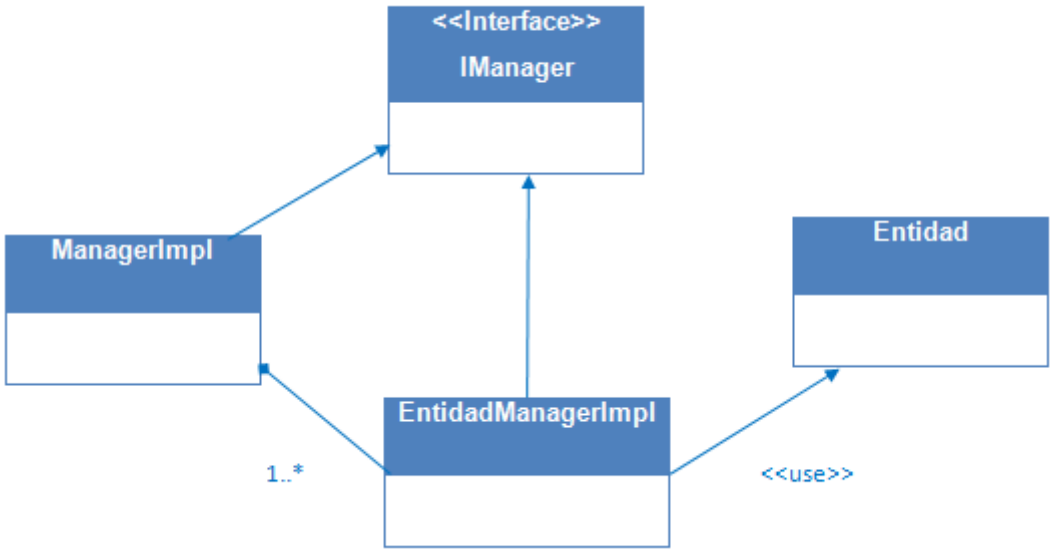

Figura 9: Capa de Negocio

2.5.4.4 Capa de Acceso a Datos (Berkeley DB)

La capa de acceso a datos mostrada es un nivel lógico que facilita el acceso a los datos y permite que todo el mundo se conecte de la misma forma, y facilitando además el mantenimiento de la aplicación. Esta capa está compuesta por los Objetos de Acceso a Datos.

Dentro de los Objetos de Acceso a Datos se encapsula la lógica de acceso a los datos, de esta manera se centraliza dicha funcionalidad.

Todo el acceso a los datos es a través del API, ofreciendo escalabilidad, alto desempeño,

transacciones protegidas.

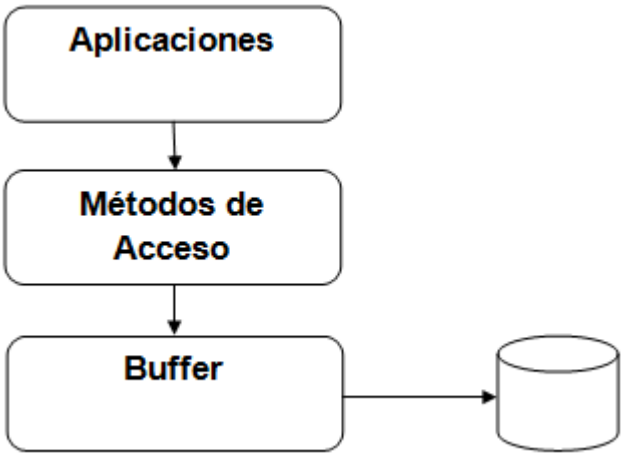

Figura 10: Arquitectura Berkeley DB

Cada capa brinda servicios a la capa superior a ella y actúa como cliente para la capa inferior.

2.6 Seguridad

La seguridad es un aspecto fundamental en una aplicación para la protección de la integridad y privacidad de la información. Esta debe implementarse usando mecanismos eficientes y probados, tanto de autenticación, como autorización y validación de los datos para ello se definió usar la estructura y métodos que tiene el software de réplica Reko garantizando:

- Seguridad de la conexión con el servidor de mensajería.
- Seguridad de encriptación de ficheros.
- Seguridad de la autenticación y la autorización web.
- Seguridad de la conexión con el web service (servicio web).

2.7 Modelo de despliegue

El modelo de despliegue de RekoFile está formado por servidores y PCs clientes como dispositivos para el funcionamiento de la aplicación tal y como es mostrado en la figura 11.

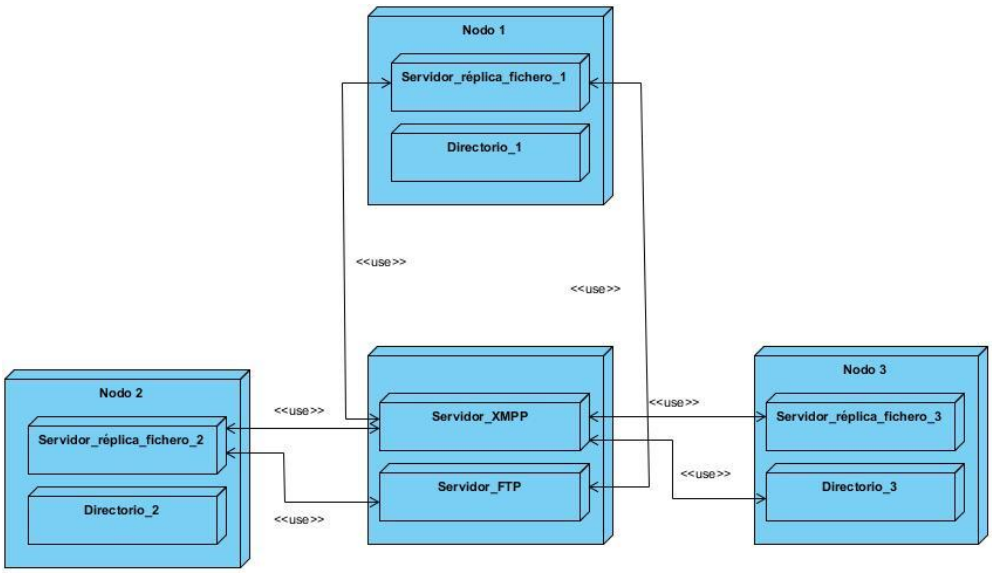

Figura 11: Modelo de despliegue

La aplicación será desplegada de la siguiente manera: varios clientes (Nodo 1, Nodo 2, Nodo 3) que podrán estar en cualquier parte con estado y conexión bidireccional al servidor de mensajería XMPP o en el caso de archivos mayores de 5 Mb por el servidor FTP.

Las PCs clientes deberán tener acceso, vía web, a los servidores y deberán tener instalado un navegador web para poder hacer uso de la aplicación.

El servidor ProFTP a través del protocolo de comunicación FTP es el encargado de gestionar los envíos entre el cliente y los servidores mayores de 5 Mb.

El protocolo XMPP se utiliza principalmente para la mensajería instantánea.

De esta forma, se garantizará el funcionamiento del componente, y el despliegue del mismo en cualquier entorno que cuente con una red local o acceso a Internet.

2.8 Propuesta de herramientas:

2.8.1 Herramienta CASE

Las herramientas case son muy utilizadas en la actualidad porque son aplicaciones que ayudan a aumentar la productividad del desarrollo de software reduciendo costes, tiempo y dinero de los sistemas en confección. El uso de las herramientas CASE presenta para el usuario que las usa varias ventajas entre las que se encuentra que ayuda a mejorar la

calidad del software.

Dentro de las herramientas CASE tenemos:

Rational Rose

Rational es una herramienta CASE basada en UML que permite crear los diagramas que se van generando durante el proceso de ingeniería en el desarrollo del software. Es completamente compatible con la metodología RUP, brinda muchas facilidades en la generación de la documentación del software que se está desarrollando, además posee un gran número de estereotipos predefinidos que facilitan el proceso de modelación del software. Es capaz de generar el código fuente de las clases definidas en el flujo de trabajo de diseño, pero tiene la limitación de que aún hay varios lenguajes de programación que no soporta o que sólo lo hace a medias. Por otra parte, una vez que se tiene el diagrama de clases persistentes a partir del cual se genera la base de datos del sistema, no existe la posibilidad de exportar ese modelo hacia algún sistema gestor de bases de datos.

Visual Paradigm

Visual Paradigm es una herramienta CASE multiplataforma que utiliza UML como lenguaje de modelado y facilita enormemente el desarrollo de software. Entre sus principales características está el soporte a ingeniería inversa, generación de código para varios lenguajes, importación desde la herramienta Rational Rose, interoperabilidad con otras herramientas a través del intercambio de información mediante exportación/importación de XML y XMI, posee un generador de informes y editor de figuras. Tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de base de datos a partir del esquema de clases. Está disponible en varias ediciones, cada una destinada a diferentes necesidades: Enterprise, Professional, Community, Standard, Modeler y Personal; posee una interfaz amigable y profesional, y se puede modelar en varios idiomas.
Visual Paradigm se puede integrar con SVN, que es un sistema de control de versiones, permitiendo el trabajo colaborativo, lo que posibilita que múltiples usuarios trabajen sobre el mismo proyecto, genera la documentación del mismo automáticamente en varios formatos como web o pdf y se integra a Entornos de Desarrollo Integrados (IDE por sus siglas en Inglés) como Eclipse o Visual Studio. Es posible crear plantillas para las especificaciones de casos de uso y describirlos, eliminando la necesidad de utilizar una herramienta externa como editor de texto.
Existen otras herramientas CASE a parte de las antes mencionadas pero la tomada en

cuenta para el trabajo arquitectónico fue Visual Paradigm. Además de su potencia y de utilizar como lenguaje de modelado UML, permite diseñar un producto con calidad y de forma rápida. Por sus características y por contar la Universidad con la licencia para su uso, se considera la más adecuada para la modelación de las vistas de la arquitectura y para los demás artefactos generados en el proyecto.

2.8.2 Ambiente de desarrollo.

Un entorno de desarrollo integrado o en inglés Integrated Development Environment (IDE) es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI.

Eclipse IDE

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

2.8.3 Framework

Un Framework no es más que “una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que puede añadírsele las últimas piezas para construir una aplicación concreta”
Los principales objetivos de cada framework consisten en: Acelerar el proceso de desarrollo. Reutilizar código ya existente. Promover buenas prácticas de desarrollo como el uso de patrones.

Spring

Es un framework de código abierto (Open Source), desarrollado por la compañía Interface 2, para aplicaciones escritas en el lenguaje Java. Los desarrolladores de este framework, lograron combinar en él herramientas tales como: Java 2 Enterprise Editions (J2EE),

incluyendo Enterprise JavaBeans (EJB), y Java Server Pages (JSP); esto facilitó brindar una estructura mucho más sólida y brindar un mejor soporte.

Este framework no necesita de muchos recursos para ser ejecutado. Está basado en inyección de dependencias (DI) y orientación a aspectos. Brinda abundante funcionalidad de infraestructura (integración con frameworks de persistencia, gestión de transacciones, etc.) y deja el desarrollo de la lógica de la aplicación al desarrollador. Contiene y gestiona el ciclo de vida y configuración de objetos de aplicación; por estos motivos es considerado un contenedor.

2.8.4 Frameworks JavaScript

En la actualidad existen muchos frameworks en JavaScript y es muy difícil determinar cuál es mejor. Es importante recordar que estos frameworks son simples herramientas que ayudan a realizar diferentes tareas de diferentes tipos, todos basados en el mismo lenguaje, javascript. Lo correcto es seleccionar uno u otro dependiendo la utilidad y la capacidad de saber cómo utilizarlo.

Entre los populares Frameworks para Javascript se tienen jQuery, Prototype, MooTools, ExtJS, Dojo y otros muchos más.

Se determinó utilizar Dojo para la presentación de interfaz de usuario y la visualización de los gráficos que actualicen los datos en tiempo real.

Dojo es otro Framework en JavaScript el cual permite añadir características dinámicas fácilmente a las páginas Web. Puedes utilizar los componentes que incorpora Dojo para mejorar la usabilidad y funcionabilidad. Se puede construir interfaces con degradados de color, widges y transacciones animadas de forma rápida y sencilla. Contiene varias características que trabajan a través de la mayoría de los navegadores, tales como: Menúes, Tabs, Tooltips y Tablas ordenables.

2.8.5 Protocolo de mensajería

Extensible Messaging and Presence Protocol **(XMPP):** es una tecnología para la comunicación en tiempo real. Proporciona una vía para enviar piezas pequeñas de XML de una entidad a otra en tiempo real. Un servicio es una característica o función que puede ser usada por cualquier aplicación. XMPP típicamente proporciona los siguientes servicios:

- **Codificación de canal:** Este servicio, brinda encriptación de las conexiones entre un cliente y un servidor, o dos servidores para mayor seguridad de las aplicaciones.[21]

- **Autenticación:** Este servicio, es definido para desarrollo de aplicaciones seguras. En este caso, el servicio de autenticación asegura que los entes intentando comunicarse deben ser primero autenticados por un servidor, que actúa como cierto portero para acceso de red.

- **Presencia:** La función de este servicio, es descubrir sobre la disponibilidad de red de otros entes. Un servicio de presencia responde la pregunta, ¿Está la entidad en línea y disponible para la comunicación u offline y no disponible? Los datos de presencia pueden también incluir información más detallada tal como si una persona está en una reunión. Típicamente, el compartir información de presencia está basado en un sistema de subscripción entre dos entes a fin de proteger la privacidad del usuario.[22]

- **One-to-one messaging:** El propósito de este servicio es enviar mensajes a otra entidad. El uso clásico de uno a uno enviando mensajes que pueden ser arbitrarios XML, y cualesquiera dos entes en una red pueden intercambiar mensajes, ellos pueden ser servidores, componentes, dispositivos, servicios de red de XMPP habilitado, o cualquiera otra entidad de XMPP.

Existen otros servicios no menos importantes que para su profundización pueden ser analizados en “XMPP: The Definitive Guide”. [23]

Una de las principales ventajas de este protocolo es que, a diferencia de otros protocolos de mensajería, se trata de un estándar libre.

2.8.6 Servidor FTP

ProFTPd

Es un software servidor FTP seguro y estable, siempre y cuando sea configurado correctamente, con licencia GPL y cuenta además, con una robusta documentación. Utilizado de forma opcional en el funcionamiento del software. Puede ser ejecutado como un demonio propio o como un servicio. Es capaz de trabajar sobre IPv6; posee un diseño modular, lo cual le permite escribir extensiones como cifrado SSL/TLS, RADIUS, LDAP o SQL como módulo. Funciona sobre sistemas operativos tales como: Linux for IBM S/390,

Linux, Solaris, Gentoo entre otros. Será utilizado para probar las implementaciones realizadas en el módulo para la trasmisión de datos de gran tamaño.

2.8.7 Lenguaje de programación

Uno de los aspectos importantes a tener en cuenta en un lenguaje de programación es la portabilidad. Poder usar la misma aplicación en distintas arquitecturas o sistemas operativos sin tener que recompilar supone un gran ahorro de desarrollo. A continuación se presentan las características más importantes del lenguaje de programación propuesto para la investigación.

Java

Ofrece toda la funcionalidad de un lenguaje potente. Es orientado a objetos, trabaja con sus datos como objetos y con interfaces a esos objetos. Está diseñado para ser lo suficientemente simple para que los programadores puedan lograr fluidez con el lenguaje. Proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando.

2.9 Base de Datos Embebidas.

Un SGBD embebido es un sistema que está estrechamente integrado con una aplicación de software que requiere acceso a los datos almacenados, de manera que el sistema de base de datos queda oculto a los fines del usuario final que usa la aplicación. Normalmente, el proceso de integración se logra mediante la compilación o enlace de alguna biblioteca junto con la aplicación de software en la cual reside, realizando la gestión de los datos a través de interfaces de programación, que pueden incluir desde SQL como lenguaje de consulta, hasta el manejo de los tipos de datos nativos del lenguaje de programación utilizado por la interfaz.

En adición a las ventajas generales de los SGBD embebidos expuestos anteriormente también se pueden mencionar algunos beneficios asociados al uso de estos:

- ✓ A diferencia de la arquitectura de comunicación cliente-servidor presente en los SGBD Relacionales, en la mayor parte de los SGBD embebidos, ambos, cliente y servidor corren juntos en el mismo proceso, reduciendo la latencia en el acceso a la base de datos, debido a que las llamadas a funciones dentro de un único proceso

son más eficientes que la comunicación entre procesos independientes. Además, se hace innecesaria o no obligatoria la necesidad de configurar una red o su administración.

- ✓ Generalmente los SGBD embebidos proveen a las aplicaciones de software que los utilizan, un API (Application Programming Interface) con todas las funcionalidades necesarias para la configuración, administración y gestión de datos. Esta característica reduce o evita completamente la necesidad de contar con personal adicional que administre o configure la base de datos, abstrayendo al usuario de todos estos procesos y limitándolo solamente a la interacción con la aplicación.

- ✓ El hecho de que este tipo de SGBD, se encuentre incrustado en la aplicación externa que lo utiliza, posibilita una mayor facilidad a la hora de desplegar el sistema, ya que no es necesario instalar el SGBD por separado.

- ✓ Los SGBD embebidos son herramientas muy configurables en aspectos relacionados con el uso de memoria, espacio en disco, nivel de concurrencia en el acceso a datos, severidad en la recuperación ante fallos, entre otras. Esto ofrece la ventaja de adecuar sus características a la de los requerimientos del sistema que lo alberga, de la forma más eficiente posible.

Estos sistemas son ideales en escenarios en los que se requiere el acceso a la base de datos por uno o muy pocos procesos, en aplicaciones en las que no se manejan grandes volúmenes de datos y se necesita un rápido acceso a los mismos, mediante consultas que no posean una elevada complejidad o algún método específico para la gestión de datos.

Otra característica a valorar a la hora de seleccionar un SGBD embebido, es su nivel de robustez; la razón de este requisito radica en que habitualmente en su entorno de funcionamiento, no existe un administrador de base de datos, quedando de parte del sistema toda la responsabilidad de respuesta ante cualquier fallo que pueda ocurrir.

En el mundo del software existen diversas bases de datos embebidas, entre ellas se pueden citar Berkeley DB, Daffodil DB, Perst, Metanotion BlockFile, Hypersonic SQL, Quadcap Embeddable Database, Gadfly, Metakit, Apache Derby, Db4o, H2, SQLite.

- ✓ **Base de Datos DB4O**

  Es un novedoso motor de bases de datos orientados a objetos de código abierto,

bajo la licencia GPL, que no requiere de administración, nativa de java.NET, y trabaja directamente con objetos. Sus siglas corresponden con la expresión base de datos para objetos, que a su vez es el nombre de la compañía que la desarrolla.

Entre sus características principales se encuentra su alto rendimiento, doble licencia: GPL (Open Source) y Comercial (que incluye soporte), por su bajo consumo de recursos, (de 600Kb a 800Kb) es especialmente apta para dispositivos móviles y entornos Clientes/Servidor, aunque no necesariamente limitada sólo a ellos, presenta un alto nivel de respuesta y participación, su documentación es clara, amplia, ordenada y orientada a ejemplos y de fácil lectura. Presenta dos modos de trabajo embebido y Cliente/Servidor. Presenta portabilidad entre .Net y Java, y transacciones ACID. Es fácil de instalar y tiene un único fichero de base de datos. Los objetos los almacenan tal y como son, y no hay que cambiar las clases para poder almacenarlas. [24]

- ✓ **SQLite**

SQLite es un Software Libre que hace que su código fuente sea de dominio público, portando una licencia GPL (General Public License). Fue creado por D. Richard Hipp, el cual implementó una pequeña librería de aproximadamente 500kb, programado en el lenguaje C.

SQLite, permite que múltiples usuarios accedan en modo escritura a la base de datos, ya que posee un mecanismo de bloqueo muy basto. Cuenta con la utilidad que permite ejecutar comandos SQL contra una base de datos SQLite en modo consola.[25]
Tiene la capacidad de reemplazar los grandes motores de Bases de Datos y acoplarse al desarrollo de nuestros proyectos informáticos, ya sea en ambientes de prototipos de sistemas como en complejos y robustos software.

Entre sus características fundamentales se encuentra que tiene una pequeña memoria y una única biblioteca para acceder a bases de datos, lo que lo hace ideal para aplicaciones de bases de datos incorporadas, realiza las operaciones de manera eficiente y es más rápido que MySQL y PostgreSQL, es portable, es compatible con ACID (Atomicity Consistency Isolation and Durability), cuenta con diferentes interfaces del API (Programming Interface), las cuales permiten trabajar con C++, PHP, Perl, Python, Ruby, groovy. No posee configuración. [25]

- ✓ **Berkeley DB**

Berkeley DB es un motor de bases de datos transaccional y escalable que puede utilizarse en cualquier aplicación, el cual provee a los desarrolladores una base de datos simple, rápida y segura, con cero administración, debido a que funciona como una biblioteca que se enlaza directamente en la aplicación eliminando la penalización en el rendimiento de los sistemas cliente-servidor y el procesamiento SQL, ideal para consultas estáticas sobre datos dinámicos. Es uno de los motores de almacenamiento soportado por MySQL, Subversión, OpenLDAP, Bogofilter y varios CMS (Content Management System). Estaba disponible con código fuente y licencia de libre distribución (free software), sin embargo fue comprado por Oracle, por lo que mantiene una licencia dual de software libre y de software privativo para implementaciones cerradas sobre Berkeley DB.

Es una base de datos incrustada con API para C, C++, Java, Perl, Python, Ruby, Tcl y muchos otros lenguajes de código abierto, que permite a los desarrolladores incorporar en sus aplicaciones un motor de base de datos transaccional, rápido y escalable, con disponibilidad y confiabilidad de clase industrial.

Permite, que los clientes y usuarios finales atengan una aplicación que simplemente funcione y administre confiablemente los datos.

Berkeley DB permite miles de hilos de control manipulando bases de datos de hasta 256 terabytes en muchos sistemas, incluidos la mayoría del tipo-UNIX y Windows, e incluso sistema operativos de tiempo real.

Es una base de datos de un rendimiento extraordinario, elimina los gastos de la comunicación interprocesos y SQL, se integra por completo en la aplicación y es invisible para los usuarios finales. Su recuperación de datos es en forma secuencial e indexada. Presenta procesos múltiples por aplicación e hilos múltiples por proceso. Contiene un Cifrado de datos por el algoritmo AES (Advanced Encryption Standard).Presenta registros de hasta 4GB y tablas de hasta 256TB.Su administración es automática y su código fuente es incluido. Disponible en los sistemas operativos: Linux, Windows, BSD Unix, Solaris, Mac OS X y tiene un soporte de los lenguajes C, C++, Java, Perl, Python, PHP, TCL y Ruby. [26]

Después de un estudio realizado de las tecnologías para la gestión de BD de forma

integrada, se decidió incluir como parte de la solución propuesta Berkeley DB, debido a que las características que presenta son las que mayores beneficios aportan al proceso de desarrollo.

Berkeley DB permite el desarrollo de soluciones personalizadas para la gestión de datos, sin los gastos indirectos asociados tradicionalmente a proyectos de este tipo. Proporciona además una colección de tecnologías de construcción en bloque de probada eficacia que se puede configurar para hacer frente a cualquier necesidad, ya sea desde una solución de almacenamiento local a servir de núcleo en la construcción de un sistema distribuido. Berkeley DB es una solución confiable con más de 15 años en producción, en productos que van desde teléfonos celulares hasta aplicaciones de comercio electrónico.

Algunos de los beneficios más importantes que se obtienen al usar Berkeley DB son:

Alto rendimiento: En adición al hecho de ser un SGBD Embebido, su gran rendimiento radica en la flexibilidad que posee para personalizar la BD a los requerimientos del sistema que lo utilice.
Alta fiabilidad y disponibilidad: Su completa semántica transaccional garantiza la integridad de los datos, recuperación ante fallos y replicación.
Cero administraciones: No requiere administradores adicionales de BD, debido a que toda la administración se realiza vía API, ocultándose al usuario final.
Múltiples modelos de datos: Se manejan distintos modelos de datos a través de las interfaces: SQL, clave/valor, Java.
SQL API: Basada en la conocida API SQLite3, Berkeley DB proporciona una interfaz para el manejo de bases de datos SQL y una herramienta para su tratamiento a través de la línea de comandos.
Soporte para los estándares ODBC y JDBC: Posibilita el acceso a las BD en disco usando manejadores para estos estándares.

2.10 Metas y restricciones arquitectónicas

Los requisitos no funcionales son aquellas propiedades que debe tener la solución y que no son una funcionalidad, como por ejemplo: alta disponibilidad, flexibilidad, interoperabilidad, mantenimiento, rendimiento, fiabilidad, reusabilidad, escalabilidad, seguridad, robustez entre otros. Dentro de las metas y restricciones arquitectónicas definidas para el componente, se encuentran los requisitos no funcionales que se enuncian a continuación.

RNF 1. Apariencia o interfaz externa

- **RNF 1.1** El sistema debe estar diseñado de forma tal que el usuario interactúe sin dificultad alguna con la aplicación, ajustándose a los estándares establecidos para el desarrollo de un buen diseño.

RNF 2. Usabilidad

- **RNF 2.1** La aplicación tiene que ser capaz de ofrecer facilidades de uso para un buen entendimiento y aceptación del producto por los usuarios finales.
- **RNF 2.2** El sistema podrá ser usado por cualquier tipo de persona que posea conocimientos básicos en el manejo de la computadora y el trabajo en la web.

RNF 3. Soporte

- **RNF 3.1** El sistema debe propiciar su mejoramiento y la incorporación de otras opciones en el futuro.

RNF 4. Portabilidad

- **RNF 4.1** El sistema debe ser ejecutado sobre cualquier sistema operativo, por sus características de ser multiplataforma.

RNF 5. Seguridad

- **RNF 5.1** El sistema debe establecer un mecanismo que permita la utilización de los servicios por usuarios autorizados.
- **RNF 5.2** Debe contar con protección contra acciones no autorizadas o que puedan afectar la integridad de los datos. Se deben garantizar comunicaciones seguras entre los clientes y el servidor.

RNF 6. Fiabilidad

- **RNF 6.1** El sistema debe ser confiable y preciso en la información que le suministra al usuario para evitar cualquier tipo de error.
- **RNF 6.2** El sistema estará disponible todo el tiempo, permitiendo el trabajo de los usuarios y las acciones de mantenimiento.

RNF 7. Ayuda

- **RNF 7.1** El sistema debe poseer un material de asistencia al usuario, en la que se explique de forma clara el uso de las opciones del sistema garantizando así el buen desempeño de los usuarios a la hora de interactuar con el mismo.

RNF 8. Disponibilidad

- **RNF 8.1** El sistema debe permanecer en constante funcionamiento, interrumpiendo el mismo solo cuando sea necesario reiniciarlo o detenerlo por tareas de mantenimiento o cambio de la configuración.

RNF 9. Software

- **RNF 9.1** Se debe disponer de cualquier sistema operativo y del navegador Mozilla Firefox 3.6.12 o superior.
- **RNF 9.2** Se debe disponer del servidor de mensajería instantánea.
- **RNF 9.3** Se debe disponer del servidor FTP ProFTPd.

RNF 10. Rendimiento

- **RNF 10.1** El intercambio de eventos y estados de la comunicación debe ser en tiempo real.

RNF 11. Hardware

- Para la puesta en práctica y desarrollo del proyecto se requieren máquinas con los siguientes requisitos:

Estaciones de Trabajo

Procesador Pentium 4 o superior.	80 GB de capacidad en disco.
1 GB de RAM.	Tarjeta de red.

2.11 Conclusiones

Este capítulo es el más importante del trabajo pues se ha descrito de manera exhaustiva la propuesta de arquitectura a aplicar en el software de réplica de archivos RekoFile, se han establecido y modelado los principales aspectos de la arquitectura como la seguridad, la interoperabilidad, la escalabilidad y la integración de cada una de las capas, descritas en cada una de las vistas, así como la etapa final de desarrollo que describe el modelo de despliegue. Se realizó un resumen de todas las tecnologías a utilizar dentro del software, que simplifiquen el trabajo de desarrollo del equipo.

Capítulo 3: Evaluación del diseño arquitectónico propuesto.

3.1 Introducción.

La calidad de un software depende en gran medida de la calidad de la arquitectura que se proponga para el mismo, para esto se hace necesario utilizar métodos existentes, teniendo en cuenta un conjunto de atributos de calidad asociados a esta propuesta.

3.2 Etapas en que se evalúa una arquitectura

La evaluación de la arquitectura se puede realizar en cualquier etapa del proceso de desarrollo, sobre todo cuando en el proyecto se comienzan a tomar decisiones que dependen de la arquitectura. Siempre se debe tener en cuenta que el costo de evaluar esa arquitectura debe ser menor que el costo de deshacer una decisión que dependa de dicha arquitectura.

Existen dos variantes que agrupan todos los momentos en que se puede evaluar una arquitectura, estas son evaluación temprana y evaluación tardía.

Evaluación temprana: En esta variante no es necesario que la arquitectura no esté completamente especificada para ser evaluada. Abarca desde las fases tempranas del desarrollo del proyecto hasta el final del mismo. Este puede sufrir cambios en la arquitectura en cualquier nivel puesto que pueden existir cambios, producto de una evaluación realizada.
Evaluación tardía: En esta se establece la evaluación cuando la arquitectura está definida y el proyecto se encuentra implementada. A este nivel es muy importante la evaluación debido a que da una medida del cumplimiento de los atributos de calidad en el sistema y da una medida de cómo será su comportamiento. [27]

3.3 Modelos de Calidad

Los modelos calidad resultan de utilidad para la predicción de confiabilidad y en la gerencia de calidad durante el proceso de desarrollo, así como para efectuar la medición del nivel de complejidad de un sistema de software. [28]

Pressman (2002) indica que los factores que afectan a la calidad del software no cambian, por lo que resulta útil el estudio de los modelos de calidad que han sido propuestos en este sentido desde los años 70.

Dado que los factores de calidad presentados para ese entonces siguen siendo válidos, se estudiarán los modelos más importantes propuestos hasta ahora: McCall (1977), Dromey (1996), FURPS (1987), ISO/IEC 9126 (1991) e ISO/IEC 9126 adaptado para arquitecturas de software, propuesto por Losavio et al. (2003).

3.3.1. Modelo ISO/IEC 9126

El estándar ISO/IEC 9126 ha sido desarrollado en un intento de identificar los atributos clave de calidad para un producto de software. Este estándar es una simplificación del Modelo de McCall e identifica seis características básicas de calidad que pueden estar presentes en cualquier producto de software.[29] El estándar provee una descomposición de las características en sub-características, que se muestran en la tabla.

Característica	Sub-característica
Funcionalidad	Adecuación Exactitud Interoperabilidad Seguridad
Confiabilidad	Madurez Tolerancia a fallas Recuperabilidad
Usabilidad	Entendibilidad Capacidad de aprendizaje Operabilidad
Eficiencia	Comportamiento en tiempo Comportamiento de recursos
Mantenibilidad	Analizabilidad Modificabilidad Estabilidad Capacidad de pruebas
Portabilidad	Adaptabilidad

	Instalabilidad Reemplazabilidad

Tabla 1. Características y sub-características de calidad – Modelo ISO/IEC 9126

Para la evaluación de la arquitectura del componente de comunicación se tomaron en cuentas los atributos de calidad relacionados directamente con la arquitectura del software: seguridad, rendimiento, funcionalidad, eficiencia, usabilidad, mantenibilidad, definidos por la ISO/IEC 9126.

3.4 Técnicas de evaluación de arquitecturas.

Según Bosch, las técnicas utilizadas para la evaluación de atributos de calidad requieren grandes esfuerzos por parte del ingeniero de software para crear especificaciones y predicciones. Estas técnicas requieren información del sistema a desarrollar que no está disponible durante el diseño arquitectónico, sino al principio del diseño detallado del sistema.

En vista de que el interés es tomar decisiones de tipo arquitectónico en las fases tempranas del desarrollo, son necesarias técnicas que requieran poca información detallada y puedan conducir a resultados relativamente precisos.

Las técnicas existentes en la actualidad para evaluar arquitecturas permiten hacer una evaluación cuantitativa sobre los atributos de calidad a nivel arquitectónico, pero se tienen pocos medios para predecir el máximo (o mínimo) teórico para las arquitecturas de software. Sin embargo, debido al costo de realizar este tipo de evaluación, en muchos casos los arquitectos de software evalúan cualitativamente, para decidir entre las alternativas de diseño.

Bosch (2000) propone diferentes técnicas de evaluación de arquitecturas de software, a saber: evaluación basada en escenarios, evaluación basada en simulación, evaluación basada en modelos matemáticos y evaluación basada en experiencia.

- **Evaluación basada en escenarios:** [30]

De acuerdo con Kazman un escenario es una breve descripción dela interacción de alguno de los involucrados en el desarrollo del sistema con éste.
Los escenarios proveen un vehículo que permite concretar y entender atributos de

calidad.

Entre las ventajas de su uso están:

- Son simples de crear y entender
- Son poco costosos y no requieren mucho entrenamiento
- Son efectivos

- **Evaluación basada en simulación:** Consiste en la implementación de componentes de la arquitectura y la implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse. Una vez implementados estos se pueden usar un conjunto de escenarios para evaluar los atributos de calidad.
- **Evaluación basada en modelos matemáticos:** Esta técnica es usada para validar atributos de calidad operables. Este puede ser combinado con la técnica de simulación donde la entrada de uno es el resultado del otro.
- **Evaluación basada en experiencia:** Esta evaluación es propiciada por arquitectos del proyecto o externos a este. Está basada en la experiencia y la intuición de estos.

Aunque la ingeniería de software basada en componentes promete mejorar altamente los niveles de calidad de los sistemas, no deja de ser importante la evaluación de la Arquitectura de Software. Sin embargo, para poder aplicar los métodos de evaluación arquitectónica tradicionales se requiere adaptarlos a la naturaleza de los componentes para poder aplicarle el método MECABIC.

Además se decidió de las técnicas estudiadas, seguir la línea de la técnica basada en escenarios con el instrumento de evaluación el Utility Tree, independientemente de que van implícitos, para evaluar arquitecturas es la más usada, como constatación del grado de cumplimiento de cierto atributo de calidad con los requerimientos del sistema. Debido a que facilita un mejor entendimiento de los atributos de calidad.

Es bueno destacar que un método de evaluación no es mejor que otro, sino que evalúa mejor, en ciertas condiciones, un atributo de calidad dado por lo que en dependencia de las condiciones y lo que se desea evaluar, será el método de evaluación empleado (que no tiene que ser solo uno).

3.5 Métodos de evaluación de arquitecturas.

Los métodos de evaluación de arquitecturas permiten identificar los conflictos entre las arquitecturas y las soluciones que se desarrollan. Existen varios métodos que ayudan

a esta identificación, entre estos se encuentran Software Architecture Analysis Method (SAAM), Architecture Trade-off Analysis Method (ATAM) y Active Review for Intermediate Designs (ARID).

3.5.1 SAAM.

El método de análisis de arquitecturas de software (SAAM, por sus siglas en inglés) es uno de los primeros que fue ampliamente promulgado y documentado. Originalmente fue creado para el análisis de modificabilidad de la arquitectura, pero ha demostrado ser muy útil para evaluar ciertos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad. [30]

Este consiste en la enumeración de escenarios que representarán los probables cambios a los que el sistema estará sometido en el futuro. Como entrada principal necesitará la descripción de la arquitectura de dicho sistema, la cual será evaluada.
La evaluación de este método está compuesta por seis pasos, los cuales son:

1. Desarrollo de escenarios.
2. Descripción de la arquitectura.
3. Clasificación y asignación de la prioridad de los escenarios.
4. Evaluación individual de los escenarios indirectos.
5. Evaluación de la interacción entre los escenarios.
6. Creación de la evaluación global.

Dependiendo del objetivo de la evaluación será el resultado de la misma. Por ejemplo, si el objetivo es evaluar una sola arquitectura, este método enumera los lugares más vulnerables a fallos dentro de la misma, en términos de los requerimientos de modificabilidad. Para el caso de que se deseen evaluar varias arquitecturas con el fin de seleccionar una que satisfaga mejor los requerimientos de calidad y con la menor cantidad de modificaciones posibles.

3.5.2 ATAM.

Según Kazman el método de análisis de acuerdos de arquitectura (Architecture Trade-off Analysis Method (ATAM), por sus siglas en inglés) está centrado en tres áreas distintas, el estilo arquitectónico, el análisis de los atributos de calidad y el método SAAM. Su nombre surge del hecho de que revela la forma específica en que una

arquitectura cumple con algunos atributos de calidad, así como la interacción de estos con otros.

El método se centra en identificar el estilo arquitectónico o enfoque arquitectónico utilizado. Estos representan los métodos aplicados en la arquitectura para cumplir con los atributos de calidad. [30]

La metodología de ATAM comprende nueve pasos divididos en cuatro fases y en nueve pasos. [31]

Fase 1: Presentación	
1. Presentación del ATAM	El líder de evaluación describe el método a los participantes, trata de establecer las expectativas y responde las preguntas propuestas
2. Presentación de las metas del negocio	Se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico.
3. Presentación de la arquitectura	El arquitecto describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio.
Fase 2: Investigación y análisis	
4. Identificación de los enfoques arquitectónicos	Estos elementos son detectados, pero no analizados.
5. Generación del Utility Tree	Se elicitan los atributos de calidad que engloban la “utilidad” del sistema (desempeño, disponibilidad, seguridad, modificabilidad, usabilidad, etc.), especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.
6. Análisis de los enfoques arquitectónicos	Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. En este paso se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance.
Fase 3: Pruebas	
7. Lluvia de ideas y establecimiento de	Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios

prioridad de escenarios.	
8. Análisis de los enfoques arquitectónicos	Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.
Fase 4: Reporte	
9. Presentación de los resultados	Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

Tabla 2. Pasos del método de evaluación ATAM

3.5.3 ARID.

El método de Análisis de diseños intermedios es conveniente para realizar revisiones parciales en etapas tempranas del desarrollo. Es conveniente saber si el diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura. ARID es un híbrido entre Active Design Review (ADR) y ATAM. ADR es utilizado para la evaluación de diseños detallados de unidades de software como los componentes o módulos. En el caso de ADR proporciona una documentación detallada del diseño y completan cuestionarios.

En el caso de ATAM se realiza una evaluación de la arquitectura en su conjunto. Por esta combinación de filosofías surge ARID para la evaluación temprana de arquitecturas de software. El método de evaluación comprende nueve pasos divididos en dos fases.

Fase 1: Actividades previas.

1. Identificación de los encargos de la revisión.
2. Preparar el informe de diseño.
3. Preparar los escenarios bases.
4. Preparar los materiales.

Fase 2: Revisión.

5. Presentación del ARID.
6. Presentación del diseño.
7. Lluvia de ideas y establecimiento de prioridad de escenarios.
8. Aplicación de los escenarios.

9. Resumen.

No se puede decir que un método es mejor que otro, en cambio sí se puede afirmar que en determinadas condiciones y para determinados atributos de calidad existen métodos que evalúan la arquitectura de manera más eficiente que otros.

3.6 Evaluación de la arquitectura. [28]

Teniendo en cuenta lo anterior expuesto y que RekoFile se encuentra en los primeros pasos del proceso de desarrollo de software se utilizará ATAM, el cual apoya a los involucrados con el proyecto a entender las consecuencias de las decisiones arquitectónicas respecto a los atributos de calidad del sistema, ya que permite realizar evaluaciones de la arquitectura, ayuda a la preparación, documentación y entendimiento de la solución. Identifica errores arquitecturales antes de la construcción del sistema. Asegura la incorporación de escenarios para la validación de la arquitectura. Desarrollo de una arquitectura más general y flexible y reduce los riesgos del proyecto.

3.7 Priorización de los escenarios de calidad

Se debe priorizar y ordenar los escenarios ya que así los arquitectos podrán contar con más orientación para tomar decisiones arquitectónicas, y los stakeholders pueden estar más conscientes de lo que esperan del sistema, y obtener una idea sobre en qué medida se va a sentir satisfecho. Una vez que se ha decidido incluir en el árbol de utilidad los escenarios más importantes, se procede a asignar un orden a los escenarios de calidad de un sistema utilizando dos criterios, a saber:

a) Evaluar el riesgo de no considerar el escenario. Se deben responder las preguntas: ¿qué pasa si este escenario no se cumple?, ¿cuántas personas se ven afectadas?, ¿es posible compensar el no responder a este escenario?

b) Evaluar el esfuerzo necesario para lograr cumplir con el escenario. En este caso se determina que cambios o integraciones de nuevos componentes se deben realizar para satisfacer el escenario. Los escenarios más difíciles son aquellos en los que se debe consumir más tiempo de análisis y el resto debe ser guardado como parte del registro.

Luego se construye el árbol de utilidad, la prioridad del escenario define en este método como un par (X, Y) en el cual X define el esfuerzo de satisfacer el escenario, y la Y indica los riesgos que se corren al excluirlos del árbol de utilidad. Los posibles valores para X e Y son A (Alto), B (Bajo) y M (Medio). El árbol de utilidad generado se toma como un plan para el resto de la evaluación que realiza el método. Indica además al equipo evaluador dónde ocupar su tiempo y dónde probar aproximaciones y riesgos arquitectónicos.

Árbol de Utilidad			
Atributo de Calidad	**Sub-característica**	**Escenario**	**Prioridad**
Funcionalidad	Seguridad	El sistema debe restringir el acceso a los datos	A
		El sistema debe aceptar los datos introducidos por el usuario solo cuando estos sean los correctos.	A
	Adecuación	El sistema cumple con los RF y RNF solicitados por el cliente.	A
	Interoperabilidad	Debe ser capaz de integrarse a otros sistemas, permitiendo la comunicación en tiempo real.	A
Fiabilidad	Tolerancia a fallos y Recuperación	Al ocurrir un error en el servicio, la operación queda cancelada por lo que se le envía un mensaje al usuario y el sistema deberá retornar a su estado inicial antes de iniciada la petición.	A
Usabilidad	Comprensibilidad, Aprendibilidad y Atractividad	El sistema debe tener la capacidad de ser atractivo, entendible para el usuario.	M
Eficiencia	Usabilidad de Recursos y	El sistema debe ser capaz de actualizar bidireccionalmente la	A

	Conectividad	información manejada durante el período en que no se mantuvo la conexión.	
	Tiempo de Respuesta	Los tiempos de respuesta a las peticiones realizadas por los usuarios deben ser en tiempo real.	A
Mantenibilidad	Cambiabilidad y Facilidad de adaptación al cambio	El sistema debe permitir que se le puedan adicionar componentes, módulos o nuevas funcionalidades.	M
Portabilidad	Mantenibilidad-Facilidad	Los componentes pueden instalarse fácilmente en todos los ambientes donde debe funcionar,	M
Leyenda: Prioridad Alta (A), Media (M), Baja (B)			

Tabla 3. Árbol de Utilidad DahBoard

ATAM es un método para identificar riegos, esto significa que se detecta áreas de riesgos potenciales dentro de la arquitectura de un complejo sistema de software.

Por lo que ATAM tiene algunas implicaciones:

- Debe ser ejecutado tempranamente en el ciclo de desarrollo del software.
- Debe ejecutarse relativamente con un bajo costo y rápido, ya que evalúa los artefactos del diseño arquitectónico.
- Produce un análisis en proporción con el nivel de detalle de la especificación de la arquitectura. Además no siempre cuando se obtiene un análisis detallado de los atributos de calidad medibles de un sistema puede ser el éxito. En cambio el éxito es lograr identificar las amenazas potenciales para el sistema.

Este último aspecto es crucial para entender los objetivos de ATAM, donde el objetivo no es predecir exactamente el comportamiento de un atributo de calidad. Esto será imposible en etapas tempranas en el escenario de diseño, porque todavía no se tiene suficiente información para hacer esta predicción. Es por ello que ATAM se centra en la identificación de riesgos. Es sumamente importante en ATAM registrar cualquier riesgo, punto sensible y puntos de intercambio.

Los riesgos son decisiones arquitecturalmente importantes, que no han sido tomadas o decisiones que han sido tomadas pero las consecuencias no han sido entendidas a plenitud.

Los puntos sensibles son parámetros en la arquitectura en los cuales la respuesta medible de algunos atributos de calidad es altamente correlacionada.

Un punto de intercambio (tradeoff) es descubierto en la arquitectura cuando un parámetro de construcción arquitectural es un anfitrión para más de un punto sensible donde los puntos de calidad medibles son afectados indistintamente por cambio en el parámetro.

Las tablas desde la cuatro hasta la doce muestran el tratamiento de los escenarios a través del método ATAM como parte del proceso de evaluación de la arquitectura. Cabe descartar la simbología utilizada en las tablas.

S es un punto sensible, como R es un riesgo, NR no riesgo y T es un punto de intercambio. Estos contendrán un número que se incrementarán en la medida que se van identificando algunas de estas características.

Escenario #: 1	**El sistema debe restringir el acceso a los datos.**			
Atributo(s)	Funcionalidad-Seguridad.			
Ambiente	Operación normal.			
Estímulo	El usuario hace uso de las funcionalidades del sistema.			
Respuesta	No hay cambios en la implementación de dicha funcionalidad.			
Decisiones arquitectónicas	Riesgo	Punto de Sensibilidad	Punto de Intercambio	No Riesgo
		S1		
Explicación	.La integridad de la Información resulta extremadamente importante en los sistemas informáticos. Para mantener la integridad de la información se restringe según el rol y los servicios instalados al usuario			

Tabla 4 Escenario #1 Atributo Seguridad

Escenario #:2	**El sistema debe aceptar los datos introducidos por el usuario**

	solo cuando estos sean correctos.			
Atributo(s)	Funcionalidad-Seguridad			
Ambiente	Operación normal.			
Estímulo	El usuario introduce datos al sistema.			
Respuesta	El sistema acepta los datos insertados correctamente, validando la entrada de los campos de la aplicación.			
Decisiones arquitectónicas	Riesgo	Punto de Sensibilidad	Punto de Intercambio	No Riesgo
		S2		
Explicación	Se lleva a cabo la validación de los formularios para alcanzar elevados niveles de confiabilidad sobre los datos. El sistema debe mostrarle un mensaje de error en el caso que el usuario introduzca los datos incorrectamente, esto le brinda la oportunidad al usuario de corregir dichos datos.			

Tabla 5 Escenario #2 Atributos Funcionalidad-Seguridad

Escenario #: 3	**El sistema cumple con los RF y RNF solicitados por el cliente**			
Atributo(s)	Funcionalidad-Adecuación			
Ambiente	Operación normal.			
Estímulo	El usuario hace uso de las funcionalidades del sistema.			
Respuesta	No hay cambios en la implementación de dicha funcionalidad.			
Decisiones arquitectónicas	Riesgo	Punto de Sensibilidad	Punto de Intercambio	No Riesgo
		S3		
Explicación	Se lleva a cabo la validación de los formularios para alcanzar elevados niveles de confiabilidad sobre los datos. El sistema debe mostrarle un mensaje de error en el caso que el usuario introduzca los datos incorrectamente, esto le brinda la oportunidad al usuario de corregir dichos datos.			

Tabla 6 Escenario #3 Atributos Funcionalidad-Adecuación

Escenario #:4	**Debe ser capaz de integrarse a otros sistemas, permitiendo la**

	comunicación en tiempo real.			
Atributo(s)	Funcionalidad, Seguridad-Interoperabilidad, Fiabilidad-Tolerancia a fallos, Eficiencia-Usabilidad de Recursos, Mantenibilidad-Estabilidad			
Ambiente	Operación normal.			
Estímulo	Necesidad de interactuar: brindar o recibir información con otros sistemas desplegados en el componente.			
Respuesta	No hay cambios en la implementación de dicha funcionalidad.			
Decisiones arquitectónicas	Riesgo	Punto de Sensibilidad	Punto de Intercambio	No Riesgo
	R1		T1	
Explicación	El sistema se concibió con la idea de que este pueda ser capaz de intercambiar flujo de datos con otros sistemas desplegados en el componente y a la hora de tomar las decisiones arquitectónicas se pensó en dicho detalle.			

Tabla 7 Escenario #4 Atributos Funcionalidad, Seguridad-Interoperabilidad, Fiabilidad-Tolerancia a fallos, Eficiencia-Usabilidad de Recursos, Mantenibilidad-Estabilidad

Escenario #: 5	**Al ocurrir un error en el servicio, la operación queda cancelada por lo que se le envía un mensaje al usuario y el sistema deberá retornar a su estado inicial antes de iniciada la petición.**			
Atributo(s)	Fiabilidad- Tolerancia a fallos – Recuperación			
Ambiente	Operación normal.			
Estímulo	Se produce un error durante la publicación-suscripción.			
Respuesta	El sistema debe ser capaz volver al estado inicial antes de realizada la petición.			
Decisiones arquitectónicas	Riesgo	Punto de Sensibilidad	Punto de Intercambio	No Riesgo
			T2	
Explicación	Lograr que el sistema retorne a su estado inicial tras la ocurrencia de algún error fue uno de los puntos en los que se tuvieron en cuenta a la hora de tomar decisiones arquitectónicas en este caso a la hora de definir los RNF, producto que influye en atributos claves			

	como la seguridad, funcionalidad y fiabilidad.

Tabla 8 Escenario #5 Atributos Fiabilidad- Tolerancia a fallos – Recuperación.

Escenario #:6	**El sistema debe tener la capacidad de ser atractivo, entendible para el usuario.**			
Atributo(s)	Usabilidad-Comprensibilidad-Aprendibilidad-Atractividad			
Ambiente	Operación normal.			
Estímulo	El usuario hace uso de las funcionalidades del sistema.			
Respuesta	No hay cambios en la implementación de dicha funcionalidad.			
Decisiones arquitectónicas	Riesgo	Punto de Sensibilidad	Punto de Intercambio	No Riesgo
			T3	
Explicación	La utilización del frameworks Dojo permitirá obtener aplicaciones entendibles y atractivas para el usuario.			

Tabla 9 Escenario #6 Atributos Usabilidad-Comprensibilidad-Aprendibilidad-Atractividad.

Escenario #: 7	**El sistema debe ser capaz de actualizar bidireccionalmente la información manejada durante el período en que no se mantuvo la conexión.**			
Atributo(s)	Eficiencia-Usabilidad de Recursos, Funcionalidad-Adecuación.			
Ambiente	Operación normal.			
Estímulo	Un período determinado de tiempo sin conexión			
Respuesta	No hay cambios en la implementación de dicha funcionalidad.			
Decisiones arquitectónicas	Riesgo	Punto de Sensibilidad	Punto de Intercambio	No Riesgo
	R2		T4	
Explicación	El sistema debe ser capaz de seguir brindando prestaciones al usuario aunque este haya perdido la conexión, una vez restablecido todo y vuelto a la normalidad la aplicación debe permitir realizar la actualización de toda la información manejadas durante ese período de tiempo.			

Tabla 10 Escenario #7 Atributos Eficiencia-Usabilidad de Recursos

Escenario #:8	**Los tiempos de respuesta a las peticiones realizadas por los usuarios deben ser en tiempo real.**			
Atributo(s)	Eficiencia, Rendimiento-Tiempo de Respuesta.			
Ambiente	Operación normal.			
Estímulo	El usuario necesita visualizar los datos en tiempo real.			
Respuesta	No hay cambios en la implementación de dicha funcionalidad.			
Decisiones arquitectónicas	Riesgo	Punto de Sensibilidad	Punto de Intercambio	No Riesgo
			T5	
Explicación	El sistema brinda sus funcionalidades a un número considerable de usuarios y debe hacerlo con un elevado rendimiento y optimización.			

Tabla 11 Escenario #8 Atributos Eficiencia, Rendimiento-Tiempo de Respuesta.

Escenario #: 9	**El sistema debe permitir que se le puedan adicionar componentes, módulos o nuevas funcionalidades.**			
Atributo(s)	Mantenibilidad-Cambiabilidad, Configurabilidad.			
Ambiente	Operación normal.			
Estímulo	Necesidad de adicionar funcionalidades al sistema.			
Respuesta	La arquitectura definida soporta la incorporación de nuevas funcionalidades, módulos al sistema.			
Decisiones arquitectónicas	Riesgo	Punto de Sensibilidad	Punto de Intercambio	No Riesgo
		S4		
Explicación	La arquitectura diseña es capaz de soportar en ambos sistemas la posibilidad de adicionarle otros módulos e incluso funcionalidades a los módulos existentes. Al hablar de configurabilidad, se hace referencia a la posibilidad que tiene el usuario de realizar cambios en el sistema, de manera que pueda adaptarlo a sus necesidades.			

Tabla 12 Escenario #9 Atributos Mantenibilidad-Cambiabilidad, Configurabilidad

Escenario #: 10	**Los componentes pueden instalarse fácilmente en todos los**

	ambientes donde debe funcionar.			
Atributo(s)	Mantenibilidad-Facilidad de adaptación al cambio, Portabilidad			
Ambiente	Operación normal.			
Estímulo	Despliegue de las aplicaciones.			
Respuesta	La arquitectura definida soporta el despliegue en cualquier sistema operativo. Se escogió como lenguaje de programación Java y el servidor de mensajería ActiveMQ para manejar los eventos.			
Decisiones arquitectónicas	Riesgo	Punto de Sensibilidad	Punto de Intercambio	No Riesgo
		S5		
Explicación				

Tabla 13 Escenario #10 Atributos Mantenibilidad-Facilidad de adaptación al cambio, Portabilidad

3.8 Toma de decisiones

Se identificaron dos riesgos. Una vez identificados los riesgos presentes en la arquitectura resulta necesario efectuar la toma de decisiones por parte de los stakeholders y el equipo que intervino en el proceso de ejecución de las pruebas de concepto de la arquitectura: arquitecto de software, líder de proyecto y algunos desarrolladores.

La toma de decisiones se ejecuta con el objetivo de mitigar los riesgos en la arquitectura desde el punto de vista de los atributos de calidad analizados y dar paso a la construcción del sistema. Como principales decisiones a tomar se aprobaron: definir políticas para la interacción del sistema con otras aplicaciones, así como para el proceso de actualización bidireccional con el objetivo de mantener la seguridad e integridad de los datos que se manejan; desarrollar un proceso de auditoría y control al diseño arquitectónico del sistema, así como analizar las consecuencias que pueden arrojar cambios significativos en las decisiones arquitectónicas definidas.

3.9 Conclusiones

En el presente capítulo se han analizado los principales elementos relacionados con la evaluación de la arquitectura de software diseñada, propiciando una correcta y

adecuada selección de seis atributos de calidad definidos por la ISO/IEC 9126: funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad; todos relacionados estrechamente con la arquitectura de software.

Se aplicó el método de evaluación de la arquitectura ATAM como parte del proceso de evaluación temprana, además de aplicar la técnica basada en escenarios y el procedimiento del árbol de utilidad.

De un total de diez escenarios clasificados con una prioridad entre alta, media y baja, el método arrojó finalmente cinco puntos de sensibilidad, dos puntos de riesgos, cinco puntos de desventajas por lo que se decidió a dar paso a la implementación del sistema. Se arribó a la conclusión de que el sistema es principalmente funcional.

Conclusiones Generales:

En este trabajo se cumplieron los objetivos trazados al inicio. Se trataron temas de gran importancia para el desarrollo de la arquitectura del software de réplica, tanto desde el punto de vista estructural, las capas y niveles físicos de la arquitectura, como de aspectos de relevancia para una arquitectura de relevancia como seguridad, auditorias, interoperabilidad entre módulos, tratamiento de excepciones, escalabilidad, facilidad de mantenimiento, entre otros.

Se definieron y usaron los estilos arquitectónicos a través del estudio y la investigación para este tipo de arquitectura, se aplicación patrones para la definición y descripción de capas lógicas y niveles físicos de nuestro sistema a través de diagramas de vistas, de forma que sea posible desplegarse de manera efectiva y eficiente.

Para lograr este diseño con mejor calidad se propuso el uso de los marcos de trabajo Spring y DOJO, todos con una probada efectividad. Con vista a lograr una mayor reutilización de los diferentes subsistemas se propuso que la estructura de los mismos contuvieran todas las capas.

Para la validación de la propuesta se utilizó el método de validación de arquitecturas ATAM. Este método permitió evaluar determinados atributos de calidad, requeridos por RekoFile, como modificabilidad, reusabilidad e integrabilidad. En la evaluación realizada se llegó a la conclusión que el diseño arquitectónico propuesto permite el cumplimiento de los requerimientos de RekoFile.

Recomendaciones:

Luego de haber analizado los resultados del presente trabajo de diploma, resulta factible arribar a las siguientes recomendaciones:

Poner en práctica el diseño arquitectónico propuesto.
Refinar la arquitectura propuesta a partir de la puesta en práctica del diseño realizado.

Referencias Bibliográficas

1. **Ecured, http://www.ecured.cu/index.php/Arquitecturas_de_software**
2. **Len Bass, P.C., Rick Kazman, Software Architecture in Practice. Second Edition ed. 2003: Addison-Wesley Professional.**
3. **Krutchen, P. The 4+1 View model of Architecture. s.l.: IEEE Software, 1995.**
4. **1471-2000, I., Recommended practice for architectural description of software-intensive systems. 2000, IEEE.**
5. **Carlos Reynoso, N.K. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. 2004; Available from: http://www.willydev.net/descargas/prev/Estiloypatron.pdf.**
6. **Shaw, Mary and Garlan, David. Software Architecture. Perspectives of an Emerging Discipline. s.l.: Prentice Hall, 1996.**
7. **Patencier, F.a.Z., Francois, the Definitive Guide to Symfony, ed. Apress. 2007.**
8. **Moreno Navarro, Juan José. Introducción al Software basado en Componentes.**
9. **Weitzenfeld, Alfredo. Ingeniería de software orientada a objetos.octubre de 2002.http://www.deeplunar.org/fusm/ingenieria%20de%20software/.**
10. **César de la Torre, U.Z.C., Miguel A. Ramos, Javier C. Nelson, GUÍA DE ARQUITECTURA N-CAPAS ORIENTADA AL DOMINIO CON .NET. Krasis ed. 2010. 433.**
11. **Burbeck, Steve. Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC). 1992.**
12. **Lou Torrijos, Ricard. Programación en castellano.http://www.programacion.com/java/tutorial/patrones2/8**
13. **Welicki, L.E., Meta-Especificación y Catalogación de Patrones de Software con Lenguajes de Dominio Específico y Modelos de Objetos Adaptativos: Una Vía para la Gestión del Conocimiento en la Ingeniería del Software, in Facultad de Informática. 2006, Universidad Pontificia de Salamanca: Madrid. p. 449.**
14. **Gamma, Erich, y otros. Design Patterns. Elements of Reusable Object-Oriented Software. s.l: Addison Wesley, 1995. ISBN 0-201-63361-2.**
15. **Larman, C., UML y PATRONES 1999, México.**
16. **Grady, Booch, Ivar, Jacobsony James, Rumbaugh. El Proceso Unificado**

de Desarrollo de Software. 2000.

17. **Erich Gamma, R.H., Ralph Johnson, John Vlissides, Design Patterns: Elements of reusable object-oriented software. 1995.**
18. **CBASQA-Desarrollo de Software, SQA. http://cbasqa.wordpress.com/2008/09/02/proceso-de-desarrollo-openup/.**
19. **Life Cicle (ciclo de vida) http://es.wikipedia.org/wiki/OpenUP**
20. **Larman, Craig. UML y Patrones. 1999.**
21. **RFC 3920: Extensible Messaging and Presence Protocol (XMPP). Available from: http://tools.ietf.org/html/rfc3921.**
22. **RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. Available from: http://tools.ietf.org/html/rfc3921.**
23. **Peter Saint-Andre, K.S., and Remko Tronçon, XMPP: The Definitive Guide. First ed. April 2009: O'Reilly Media.**
24. **Db4objects. http://www.db4o.com/espanol/.**
25. **SQLite. http://www.sqlite.org/.**
26. **Oracle. Oracle. http://www.oracle.com/technology/products/berkeley-db/index.html.**
27. **Bosh, J. 2000. Design & Use of software architecture. s.l.: Addison Wesley, 2000.**
28. **http://es.scribd.com/doc/57257044/Guia-Arquitectura-v-2**
29. **Pressman R. (2002) Ingeniería de Software. Un Enfoque Práctico. Quinta Edición. McGraw Hill**
30. **R. Kazmar, P. Clement, M. Cleint. 2001. Evaluating Software architectrues. Methods and cases studies. s.l.: Addison Wesley, 2001.**
31. **http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm**