Universidad de las Ciencias Informáticas.

Facultad 2.

Trabajo de Diploma para optar por el título de

Ingenieros en Ciencias Informáticas.

Título: Arquitectura del Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles.

Autor(es): *Susana Fuentes Jiménez*

Christian Delgado Quintero

Tutor(es): *Ing. José Antonio Plá Rodríguez*

Co-Tutor: *Ing. Vladimir Milián Núñez.*

Ciudad de la Habana

Junio de 2009

Declaración de autoría

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

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

Autores:

_____

Susana Fuentes Jiménez

Tutor:

_____

Ing. José Antonio Pla Rodríguez

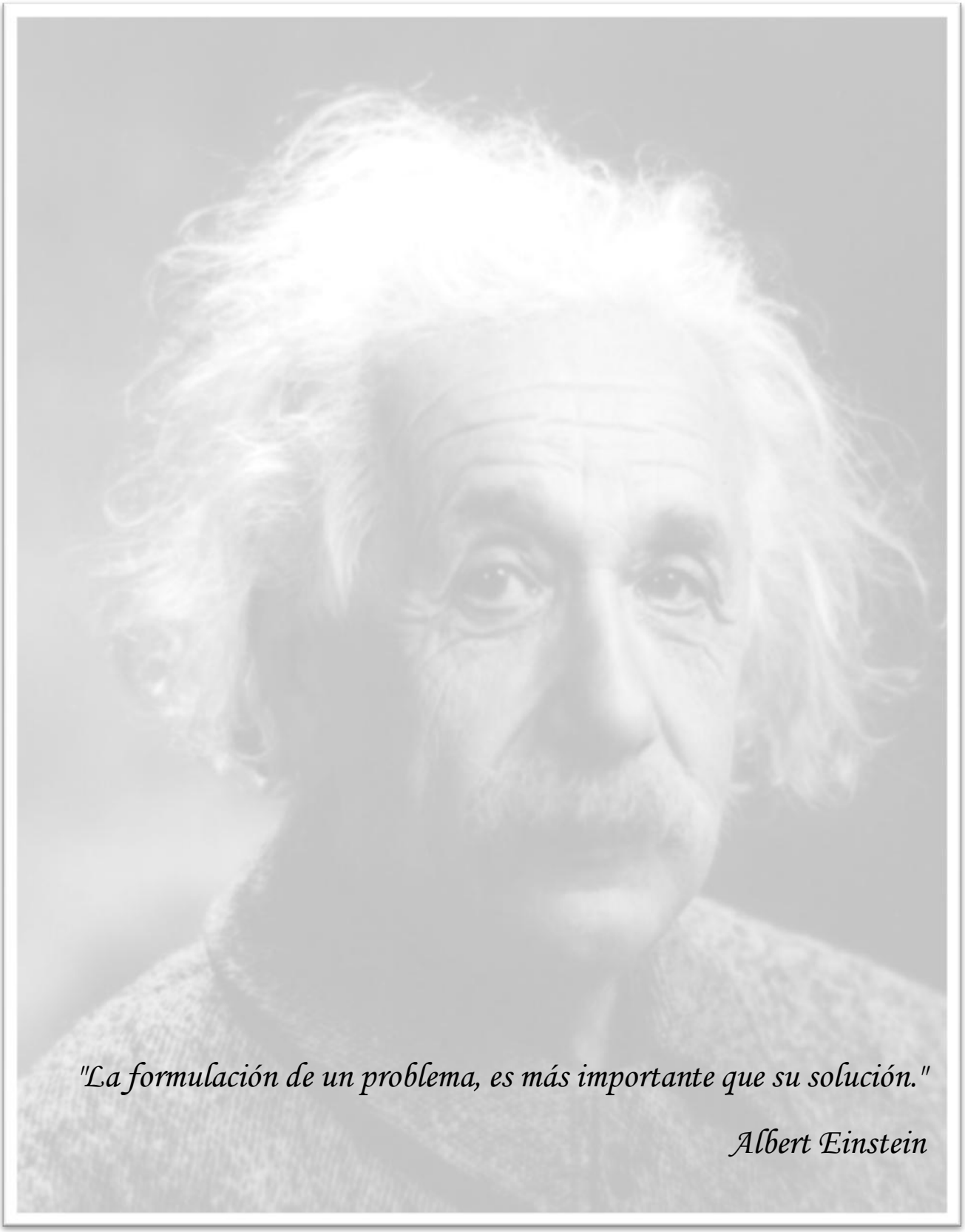
Co-Tutor:

_____

Christian Delgado Quintero

_____

Ing. Vladimir Milián Núñez



"La formulación de un problema, es más importante que su solución."

Albert Einstein

Agradecimientos

De Susana:

A mis padres por ser mis amigos por encima de todo, mis guías, y estar siempre a mi lado brindando luz y estabilidad.
A Vladimir, por su apoyo (emocional e intelectual), amor y confianza.
A mi familia que siempre me ha apoyado en todo y me ha brindado la fuerza para seguir.
A mi compañero de tesis, Christian por coincidir con mi manera de pensar y por trabajar duro por la realización de este trabajo.
A mis amigos (Abdel, Abel, Isabel, Suammy, Javier, Albin, Antonio, Miguel y Daymí.)
A mi profesora Aymée Hernández, quien ayudó mucho en mi formación.
A Pla, mi tutor.
A la Revolución, por haberme dado la posibilidad de realizar mis estudios y obtener los conocimientos necesarios para realizar este trabajo.
Gracias a todos!!!

De Christian:

A mi madre Ute Elizabeth, por ser mi guía, la inspiración de mi vida, mi amiga, mi apoyo moral, mi faro, mi refugio, la fuente de mis fuerzas y el origen de mis sueños, por su incondicionalidad y su eterno amor.
A mi padre Emilio, por su ejemplo, por querer que fuese alguien en la vida, por sus consejos y por su plena confianza en mí.
A mi hermano Manuel, por ser mi apoyo, por su confianza y por ayudarme en muchos momentos.
A mi novia Mylen, por su cariño, paciencia, amor y comprensión.
A mis abuelos, que a pesar de la distancia sé que están muy orgullosos de este momento.
A la gran familia que tengo, que forma parte de mi ser y me han apoyado en todos estos años.
A Susana, por ser mi amiga, mi compañera de Tesis, y por ayudarme a ser un mejor ingeniero.
A Pla, mi tutor.
A mis compañeros y hermanos de Universidad (Reyner, René, Luis, Yadier, Alejandro, Yandrey).
A todos mis amigos por estar ahí siempre, y soportarme.
A la UCI por haber formado una mejor persona tanto moral como profesionalmente.
A todos y a cada uno les doy las gracias!!!!!!

Dedicatoria

De Susana:

A mis padres, por todo y más.

A Vlady, por ser mi sol.

A mi familia, por su constante cariño.

De Christian:

A mis padres y hermano, por estar siempre presente.

A Mylen, por su sonrisa.

A mi familia, por el apoyo.

Resumen

Cubacel[1], se ha visto en la necesidad de extender sus prestaciones. De ahí que posibilite: la venta de contenidos a los teléfonos de sus usuarios, información actualizada con noticias nacionales, internacionales, eventos deportivos y partes meteorológicos, la promoción de productos y servicios; así como, la disponibilidad del servicio de páginas amarillas; posibilidades que se hacen realidad con la construcción de una plataforma de gestión de contenidos para dispositivos móviles, la creación de un nuevo portal *web* para la administración, y la actualización del portal WAP para los servicios restantes.

La plataforma debe convertirse en un producto comercializable para la Universidad de las Ciencias Informáticas (UCI). Para tal fin, la arquitectura aseguró que dicha plataforma pueda ser utilizada por cualquier operador de telefonía móvil en el mundo, con solo atender a pequeñas adaptaciones, de acuerdo a las características específicas de cada cual.

El presente trabajo muestra la arquitectura para el Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles. Se analizan las principales tendencias, tecnologías y metodologías actuales, se precisan los requerimientos del sistema; y se definen las herramientas que deben ser usadas por el equipo de trabajo para la implementación y puesta en marcha de la aplicación.

[1] Cubacel, es la entidad cubana que presta servicios públicos de telefonía celular.

Índice

Introducción

Los cambios tecnológicos en el campo de la informática hacen que esta ciencia sea cada día más dinámica. Tecnologías como las redes, el comercio electrónico, el teletrabajo y la telefonía, están teniendo tal repercusión en el mundo, que cada vez resulta más difícil sustraerse a esta corriente de innovación.

Cuba se propuso, tras el Triunfo Revolucionario de 1959, un camino de desarrollo que pudiera solucionar por igual las necesidades materiales básicas y espirituales de su población. Así, se diseña e inicia la aplicación de estrategias que permiten convertir los conocimientos y las tecnologías de la información y las comunicaciones, en instrumentos a disposición del avance de nuestro país.

De este modo, durante los primeros años del presente siglo, Cuba se ha centrado en la informatización de los procesos de la sociedad; lo que ha llevado a crear sistemas eficientes, a la altura de las necesidades de los usuarios, que se van introduciendo como parte de los avances económicos que va teniendo el país.

Uno de los ejemplos lo constituye la telefonía y, dentro de ella, la móvil - también llamada telefonía celular-. Es una de las ramas de las telecomunicaciones que presenta un desarrollo más acelerado y una mayor aceptación por parte de los usuarios.

El teléfono celular o móvil es un dispositivo inalámbrico electrónico que permite tener acceso a la red de telefonía. Su característica más importante es la movilidad, pues permite comunicarse desde casi cualquier lugar. Aunque su principal función es la comunicación de voz, como el teléfono convencional, su rápido desarrollo ha incorporado otras funciones como son mensajes cortos de texto y multimedia, cámara fotográfica, agenda electrónica, acceso a Internet, reproducción de video y música, e incluso GPS (del inglés *Global Positioning System, Sistema de Posicionamiento Global*).

Otro dispositivo inalámbrico electrónico que permite el acceso a la red de telefonía es el PDA (del Inglés *Personal Digital Assistant*, Asistente Personal Digital), el cual se clasifica como una computadora de mano. Sus prestaciones fundamentales son: navegación por internet, correo electrónico, juegos, música, películas, creación de documentos, calendarios, lista de contactos y recordatorios; además de todas las funciones de los teléfonos celulares.

Para la realización de este trabajo se define como dispositivo móvil el conjunto formado por los teléfonos celulares y los PDA.

En consecuencia con el avance que ha tenido la telefonía celular en el mundo y la liberación de venta de teléfonos móviles en Cuba, Cubacel se ve precisada a aumentar y mejorar los servicios que, de este campo, se ofrecen en el país.

Aunque dicha empresa cuenta ya con un portal WAP (del inglés *Wireless Application Protocol*), este no cumple con todas las necesidades que el desarrollo de la tecnología móvil y la sociedad imponen. Por consiguiente, se pretende extender la gama de servicios que hoy en día se ofrecen.

Entre los nuevos servicios podremos encontrar: la posibilidad de la descarga de contenidos (tonos, logos y melodías) hacia los móviles de los usuarios; información sobre noticias nacionales e internacionales, eventos deportivos y partes meteorológicos; ayuda de páginas amarillas; así como la promoción de productos y servicios de la empresa en cuestión a través de este portal.

Esta problemática nos indica que el **problema científico** radica en cómo desarrollar una arquitectura robusta, flexible y reusable para el Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles. El **objeto de estudio** son los procesos asociados a la gestión y entrega de contenidos para dispositivos móviles. El **campo de acción** son los estándares vinculados a la entrega de contenidos para dispositivos móviles. Como **objetivo principal** se plantea la definición de una arquitectura de *software* para el núcleo de Gina[2] que permita tomar decisiones significativas sobre la organización de dicha plataforma, y como **objetivo específico,** describir la arquitectura del sistema a través de las diferentes vistas arquitectónicas que recomienda la metodología utilizada. Para darle cumplimiento a dichos objetivos se plantean las siguientes tareas:

- Estudio detallado de los procesos de gestión y entrega de contenidos para dispositivos móviles.
- Determinar los estilos y patrones arquitectónicos que se emplearán durante el proceso de desarrollo y diseño del sistema.
- Definir las herramientas y las tecnologías a utilizar para la implementación de la plataforma.

La arquitectura del Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles, fue diseñada de manera independiente a la forma que se utilice para mostrar los contenidos a los clientes, con lo que se le transforma así en un producto UCI, que pueda comercializarse con pequeños cambios según los requerimientos de los nuevos clientes.

En este trabajo se presenta la arquitectura para la construcción del Núcleo de Gina, se describen los principales artefactos generados que sugiere la metodología de desarrollo de *software* seleccionada, la visión general arquitectónica y la definición de las herramientas y tecnologías necesarias para el desarrollo e implementación de dicha arquitectura.

[2] Gina: Nombre oficial de la Plataforma de Gestión de Contenidos para Dispositivos Móviles.

El presente documento está estructurado en 4 capítulos:

Capítulo 1 Fundamentación teórica: Está dirigido al estudio de la arquitectura de s*oftware* y a la selección de las herramientas de modelación y descripción del sistema.

Capítulo 2 Línea base de la arquitectura: Se describen las características fundamentales del Núcleo como las concepciones generales, el estilo arquitectónico, herramientas y tecnologías de desarrollo.

Capítulo 3 Descripción de la arquitectura: Describe la arquitectura del Núcleo, sus componentes y las vistas arquitectónicas de casos de uso, lógica, implementación, despliegue y datos.

Capítulo 4 Evaluación de la arquitectura: Valora cómo la arquitectura del Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles cumple con los atributos de calidad.

Capítulo 1: Fundamentación teórica

1.1 Introducción

El presente capítulo está dirigido al estudio de la arquitectura de s*oftware*; se explica qué es, cómo surge y los objetivos que debe cumplir. Así mismo, se analizan diferentes metodologías de desarrollo, estilos arquitectónicos, patrones, herramientas CASE y lenguajes de descripción arquitectónica a fin de seleccionar, en cada caso, los más adecuados para el Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles.

1.2 La arquitectura de *software*

La arquitectura de s*oftware* es la organización fundamental de un sistema que se manifiesta en sus componentes y en las relaciones que se establecen entre ellos; así como, en el ambiente y los principios que orientan su diseño y evolución (1). Aunque también se define como una descripción de los subsistemas y componentes de un sistema informático y las relaciones entre ellos (2).

La arquitectura de *software* abarca decisiones importantes sobre:

- La organización del sistema *software*.
- Los elementos estructurales que compondrán el sistema y sus interfaces.
- El comportamiento de los elementos en subsistemas progresivamente más grandes.
- El estilo de la arquitectura que guía esta organización: los elementos y sus interfaces, sus colaboraciones y su composición.

1.2.1 Orígenes de la arquitectura de *software*

Al surgir la informática, la construcción de *software* era desorganizada y se basaba solo en la implementación. Es en 1968 cuando Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda, propuso que se estableciera una estructuración correcta de los sistemas de *software* antes de lanzarse a programar. Dijkstra sostenía que las ciencias de la computación eran una rama aplicada de las matemáticas y sugería seguir pasos formales para descomponer problemas mayores (3).

En 1975, Frederick Phillips Brooks Jr., diseñador del sistema operativo OS/360, utilizaba el concepto de arquitectura del sistema para designar "la especificación completa y detallada de la interfaz de usuario". Consideraba que el arquitecto es un agente del usuario, igual que lo es quien diseña su casa (4), y además, distinguía entre arquitectura e implementación; mientras que la primera decía "qué hacer", la segunda se ocupaba del "cómo hacerlo".

En 1992, la arquitectura de *software se* define como disciplina en la publicación "Foundations for the study of software architecture" escrito por Dewayne Perry y Alexander Wolf. En 1996, Shaw y Garlan en su libro *Software Architecture. Perspectives of an Emerging Discipline* plantean que en esta disciplina resultan muy importantes los elementos como: los protocolos para la comunicación, la sincronización, y el acceso a los datos; la asignación de la funcionalidad para diseñar elementos; la distribución física; la composición de los elementos del diseño; el escalamiento y el funcionamiento; y la selección entre alternativas del diseño" (5).

1.2.2 Objetivos más importantes que debe cubrir un diseño de arquitectura

A continuación se explican tres de los objetivos que no deben faltar en un diseño arquitectónico.

- Dar cobertura a la funcionalidad: El diseño del *software* debe priorizar la inclusión de todas las funcionalidades que se especifican, de manera tal que en él se conciba (en su diseño) la solución de los requisitos que le dan funcionalidad al sistema.

- Facilitar el desarrollo del proyecto de software: Un *software* puede ser un proyecto complejo en el que intervengan decenas de personas y que se extienda durante años. Será un objetivo crucial que el reparto de funciones entre los distintos módulos posibilite un desarrollo a un mismo tiempo, independiente y coordinado.

- Optimizar el uso de los recursos del hardware: Es necesario conocer las capacidades del *hardware* (los procesadores, la memoria, el almacenamiento, la capacidad de generación gráfica, la capacidad de comunicaciones) para explotarlos al máximo.

La arquitectura dirige el desarrollo de los sistemas de *software*. Debe garantizar que se cumpla con los requisitos funcionales y no funcionales de los clientes. Se refina durante todo el ciclo de vida de los proyectos y contribuye a que estos no excedan los límites establecidos de costo y tiempo.

1.3 Estilos arquitectónicos

Toda arquitectura posee una manera propia: como las construcciones civiles, que todas tiene sus particularidades, pero todas mantienen una misma cualidad. Así mismo, la arquitectura de *software* presenta estilos generales para guiar la estructuración de los sistemas.

Un estilo proporciona un lenguaje de diseño con un vocabulario, a partir del cual los arquitectos pueden construir patrones de diseño para resolver problemas específicos. Los estilos se vinculan con patrones de arquitectura que ofician como microarquitecturas (2).

Otra de sus definiciones, lo concibe como un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para ensamblar un sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo (6).

Los estilos expresan esquemas de organización estructural fundamentales para los sistemas de software. Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen guías y lineamientos para organizar las relaciones entre ellos (7).

1.3.1 Familias de estilos

Estilos de Flujo de Datos
- Tubería y filtros

Estilos Centrados en Datos
- Arquitecturas de Pizarra o Repositorio

Estilos de Llamada y Retorno
- Model-View-Controller (MVC)
- Arquitecturas en Capas
- Arquitecturas Orientadas a Objetos
- Arquitecturas Basadas en Componentes

Estilos de Código Móvil
- Arquitectura de Máquinas Virtuales

Estilos heterogéneos
- Sistemas de control de procesos
- Arquitecturas Basadas en Atributos

Estilos Peer-to-Peer
- Arquitecturas Basadas en Eventos
- Arquitecturas Orientadas a Servicios
- Arquitecturas Basadas en Recursos

1.3.2 Estilos arquitectónicos más comunes

La selección del estilo arquitectónico depende en gran medida de las necesidades y el alcance del *software*, sin embargo los más comunes son: orientada a servicios, basado en componentes y dividido en capas. A continuación se explican brevemente dichos estilos.

Arquitectura Orientada a Servicios

La arquitectura SOA separa los procesos de negocio de las funciones automatizadas y organiza estas últimas en módulos individuales que se catalogan en un diccionario de servicios, facilitando su uso dentro de toda la organización.

Está formada por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal, independiente de la plataforma y del lenguaje de programación. La definición de la interfaz "encapsula" las particularidades de una implementación.

Uno de los principales objetivos que persigue este tipo de arquitecturas es el poder realizar cambios en los procesos de negocio de forma fácil. Para ello se plantea la definición de estos utilizando archivos de texto (XML) en lugar de hacerlo de forma programática dentro de la lógica del sistema.

Permite la creación de sistemas altamente escalables que reflejan el negocio de la organización y, a su vez, brinda una forma estándar de exposición e invocación de servicios (común pero no exclusivamente servicios web)(8).

Arquitectura Orientada a Componentes

El objetivo de la arquitectura orientada a componentes es construir aplicaciones complejas mediante ensamblado de componentes que han sido previamente diseñados por otras personas o por subgrupos del equipo de desarrollo, a fin de ser rehusados en múltiples aplicaciones.

La arquitectura de una aplicación, basada en componentes, consiste en la unión de algunos componentes específicos (que se diseñan particularmente para ella), y de otros previamente diseñados. El ensamblado de todos ellos proporciona los servicios que se necesitan en el sistema.

En la arquitectura orientada a componentes, la interfaz constituye el elemento básico de conexión. Cada componente debe describir de forma completa las interfaces que ofrece, así como las interfaces que requiere para su operación. Y debe operar con independencia de los mecanismos internos que utilice para soportar la funcionalidad de la interfaz.

La modularidad, la reusabilidad y la robustez son las características relevantes que abalan a la arquitectura basada en componentes, como una evolución de la tecnología orientada a objetos; permitiendo una mejora en la calidad, la disminución de los tiempos de desarrollo, los costos y un incremento en cuanto a la facilidad para gestionar la creciente complejidad de los sistemas (9).

Arquitecturas en capas

Es una organización jerárquica de modo que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior (5).

Las interacciones entre las capas ocurren generalmente por invocación de métodos: por definición, los niveles más bajos no podrán utilizar las funcionalidades ofrecidas por las capas superiores. Este estilo facilita la modularidad del sistema, la localización de errores y mejora considerablemente, el soporte del sistema; además de la escalabilidad. Asimismo, se pueden hacer cambios en una capa con pocos o ningún efecto en las demás.

1.4 Patrones

Resulta ilógico no utilizar el conocimiento adquirido y publicado por expertos a través de años de estudio. La utilización de determinados patrones significa para los desarrolladores, indistintamente de su experiencia y habilidad en el tema, un ahorro considerable en cuanto a tiempo y recursos. Permite que la experiencia acumulada en el diseño de aplicaciones pueda ser aprovechada de manera óptima; de ahí que, facilite elaborar diseños más simples, robustos y generales; factibles por ende, al cambio. Aunque sin duda, un buen diseño no debe limitarse a una aplicación concreta, ya que debe basarse en soluciones que han funcionado bien en otras ocasiones (6).

Se asume por patrón entonces: aquella solución probada, que se puede aplicar con éxito a un determinado tipo de problema, y que pudiera aparecer repetidamente en un marco específico (6).

1.4.1 Elementos esenciales de un patrón.

Todo patrón consta de un nombre y un problema, y en ocasiones, de varias soluciones y consecuencias. A continuación se describen estos elementos.

El nombre: Cada patrón popular debe ser conocido por todos los desarrolladores de *software* con un único identificador, que garantice una mayor comprensión entre todos ellos, tal como un diccionario que obligue a que todos a usen el mismo lenguaje.

El problema: Los patrones surgen para darle solución a un problema. Antes del patrón, debe existir el problema, y este último, puede repetirse en varios contextos.

La solución: Todos los patrones tienen que dar una solución al problema planteado. Puede pasar que, para un mismo patrón, haya más de una solución, quedando en manos del desarrollador elegir la más conveniente.

Las consecuencias: Puede que este elemento sea el que más atente contra los desarrolladores, pues hay que aceptar los compromisos que se tienen al aplicar patrones determinados.

1.4.2 Clasificación de los patrones

Para facilitar el estudio de los patrones, los mismos se dividen en diferentes ramas, a continuación se mencionan las familias más destacadas.

Patrones de arquitectura

Están relacionados con la interacción de objetos dentro o entre niveles; basándose en: adaptabilidad a requerimientos cambiantes, rendimiento, modularidad y acoplamiento entre otros problemas arquitectónicos.

Patrones de diseño

Estos patrones fueron construidos para darle soluciones elegantes a situaciones difíciles en el diseño (multiplicación de clases y en la adaptabilidad a requerimientos cambiantes). En esta calificación encontramos los patrones GRAPS (*General Responsability Asignment Software Patterns*) que describen los principios fundamentales de la asignación de responsabilidades a objetos y los GoF (*Gans of Four*) que enmarcan los llamados patrones de diseño estructurales, creacionales y de comportamiento.

Patrones de análisis

Los patrones de análisis son un conjunto de clases y relaciones entre ellas, que tienen algún sentido en el contexto de una aplicación. Representan una estructura que puede ser

válida para otras aplicaciones. En el contexto de las aplicaciones empresariales, dichos patrones representan el conocimiento que le da el valor a la organización.

Patrones de proceso o de organización

Ellos tratan todo lo concerniente a los procesos de administración de proyectos, técnicas, o estructuras de organización.

Patrones de idioma

Norman la nomenclatura en la que se escriben, diseñan y desarrollan los sistemas, además proporcionan vocabulario y entendimiento común.

1.5 Metodologías de desarrollo

La metodología de desarrollo es un conjunto de procedimientos, técnicas, herramientas, y soporte documental que ayuda a los desarrolladores a realizar un nuevo *software*. Si se sintetiza lo anterior se tendrá que, una metodología representa el camino para desarrollar *software* de una manera sistemática (10).

Las metodologías persiguen tres necesidades principales:

- Mejores aplicaciones.
- Un proceso de desarrollo controlado.
- Un proceso normalizado en una organización no dependiente del personal.

Los procesos se descomponen hasta el nivel de tareas o actividades elementales. Cada tarea está identificada por un procedimiento que define la forma de llevarla a cabo.

En ocasiones, el diseño de un *software* se realiza de manera rígida con los requerimientos que el cliente solicitó, y si el usuario final demanda un cambio durante la fase de prueba, el único procedimiento factible sería volver a empezar la fase de implementación; lo cual se origina por el uso de metodologías tradicionalistas o, en el peor de los casos, de ninguna. Como solución a dichos problemas se sugiere el uso de las metodologías ágiles, las cuales se adaptan fácilmente a los cambios en los requerimientos.

Algunas de las metodologías que se pueden utilizar para guiar un proceso de desarrollo de *software* son:

Proceso Unificado de Desarrollo o RUP

(RUP por sus siglas en inglés *Rational Unified Process)* está constituido por 9 flujos de trabajo (los 6 primeros son conocidos como flujos de ingeniería y los 3 últimos de apoyo): modelación del negocio, requisitos, análisis y diseño, implementación, prueba, instalación, administración de configuración y cambios, administración de proyecto y ambiente; los cuales tienen lugar sobre 4 etapas o fases: inicio, elaboración, construcción y transición. Esta metodología se diseñó principalmente para proyectos a largo plazo y establece refinamientos sucesivos de una arquitectura ejecutable (11).

Se caracteriza por ser:

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 estos describen la funcionalidad total del sistema.

Centrado en la arquitectura: 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. Es real que con la primera iteración no culmina el trabajo. En las iteraciones restantes y definidas en el plan de desarrollo del proyecto, se estructuran las demás necesidades del sistema y se refinan las ya construidas en iteraciones anteriores, con lo que se logra cada vez mayor calidad en la aplicación hasta llegar a cumplir cabalmente todos los requisitos.

Programación Extrema o XP

(XP por sus siglas en inglés *eXtreme Programing*), es una metodología ágil de desarrollo de *software* que se basa en la simplicidad, la comunicación y la retroalimentación del cliente. Su objetivo es la programación rápida y generalmente se usa en proyectos a corto plazo. XP puede hacer pruebas a los principales procesos y obtener las fallas que

pudieran ocurrir en el futuro. Se centra en la reutilización de código, para lo cual se crean patrones o modelos estándares, y de esta forma, es más flexible al cambio. Además propone la programación en pares, que consiste en la participación de dos desarrolladores en una misma estación (12).

1.6 Herramientas CASE

Las herramientas CASE (*Computer Aided Software Engineering*, Ingeniería de *Software* Asistida por Ordenador) son aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de *software* reduciendo el costo de las mismas en términos de tiempo y dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del *software* en tareas como el proceso de realización de un diseño del proyecto, cálculo de costos, generación de parte del código automáticamente con el diseño dado y documentación, entre otras.

Están generalmente enfocadas a mejorar la productividad en el desarrollo y mantenimiento del *software*, a automatizar su desarrollo, documentación y generación de código, además de que facilitan el uso de las distintas metodologías propias de la ingeniería del *software* (13).

Rational Rose

Rational Rose es una herramienta CASE que utiliza UML (por sus siglas en inglés, *Unified Modeling Language*) que permite construir los diagramas que se van necesitando durante el proceso de ingeniería según la metodología de desarrollo seleccionada. Es compatible con 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. Es capaz de generar el código fuente de las clases definidas en el flujo de trabajo de diseño (14).

Visual Paradigm

Visual Paradigm es una herramienta CASE multiplataforma que utiliza UML como lenguaje de modelado y facilita el desarrollo de *software*. Entre sus principales características están, el soporte a la ingeniería inversa, generación de código para varios lenguajes, importación desde proyectos realizados con la herramienta Rational Rose, interoperabilidad con otras herramientas a través del intercambio de información mediante exportación/importación de XML y XMI[3], así como, el contar con un generador de informes y editor de figuras.

[3] XMI: Formato que utiliza la herramienta Umbrella.

Tiene la capacidad de crear el esquema de clases a partir de una *base de datos* y viceversa. 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, y favorecer, de esta forma, el trabajo colaborativo, con lo cual se garantiza que múltiples usuarios trabajen sobre el mismo proyecto. Genera la documentación de este automáticamente en varios formatos y es compatible con Entornos de Desarrollo Integrados (IDE por sus siglas en inglés *Integrate Development Enviroment*) 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 (15).

El hecho de que la Universidad de las Ciencias Informáticas, entidad donde se desarrolla la plataforma, cuente con la licencia de esta herramienta, constituye un parámetro importante para preferirlo.

Enterprise Architect

Enterprise Architect es una herramienta de diseño y análisis que utiliza UML, y cubre el desarrollo de *software* desde los requerimientos hasta el mantenimiento. Es una herramienta multi-usuario, basada en *Windows*, diseñada para ayudar a construir *software* robusto y fácil de mantener; ofrece documentación flexible y de alta calidad.

Enterprise Architect provee trazabilidad completa desde los requisitos hasta el despliegue. Está equipada con la información que los miembros del equipo de proyecto necesitan sobre la ubicación de recursos y tareas, los equipos de administración de proyectos y la calidad para ayudarles a entregar productos en tiempo (16).

1.7 Lenguajes de descripción de arquitectura

ADL (del inglés Architecture Description Language) es el Lenguaje descriptivo de modelado que se focaliza en la estructura de alto nivel de la aplicación.

Los ADLs definen abstracciones arquitectónicas y mecanismos para descomponer un sistema en componentes y conectores, especificando de qué manera estos elementos se combinan para formar configuraciones y definiendo las familias de arquitecturas o estilos (17).

1.7.1 Historia de los ADLs

Los Lenguajes de Descripción de Arquitecturas nacen en la década de 1990 y, por ser tan jóvenes, carecen de un estándar. El uso de los que ya existen, implica el estudio de una sintaxis especializada.

Sin embargo, en lo que va del siglo XXI, se han materializado diversas propuestas para describir y razonar en términos de arquitectura de *software*, muchas de las cuales han asumido la forma de ADLs (17).

La tabla 1 muestra cronológicamente el surgimiento de los ADLs más comunes.

ADL	Fecha	Investigador - Organismo
CHAM	1990	Berry / Boudol
Rapide	1990	Luckham (Stanford)
Darwin	1991	Magee, Dulay, Eisenbach, Kramer
LILEANNA	1993	Tracz (Loral Federal)
MetaH	1993	Binns, Englehart (Honeywell)
Aesop	1994	Garlan (CMU)
ArTek	1994	Terry, Hayes-Roth, Erman (Teknowledge, DSSA)
Wright	1994	Garlan (CMU)
Acme	1995	Monroe & Garlan (CMU), Wile (USC)
SADL	1995	Moriconi, Riemenschneider (SRI)
UML	1995	Rumbaugh, Jacobson, Booch (Rational)
UniCon	1995	Shaw (CMU)
C2	1996	Taylor/Medvidovic (UCI)
Jacal	1997	Kicillof , Yankelevich (Universidad de Buenos Aires)
Armani	1998	Monroe (CMU)
xADL	2000	Medvidovic, Taylor (UCI, UCLA)

1.7.2 ADLs más utilizados

A continuación se explican brevemente los lenguajes de descripción de arquitecturas más comunes: Acme, C2, Jacal y UML; no precisamente en orden de explotación.

Acme

Acme se define como una herramienta capaz de soportar el mapeo de especificaciones arquitectónicas entre diferentes ADLs, como un lenguaje de intercambio de arquitectura. No es entonces un ADL en sentido estricto, aunque posee numerosas prestaciones que son propias de los ADLs y la literatura de referencia acostumbra a tratarlo como tal. En su sitio oficial, se reconoce que, como ADL, aún no está apto para cualquier clase de sistemas, pero puede describir sistemas “relativamente simples” (18).

C2

C2 no es estrictamente un ADL, sino un estilo de arquitectura de *software* que se ha impuesto como estándar en el modelado de sistemas que requieren un intensivo intercambio de mensajes y que suelen poseer una interfaz gráfica dominante (19).

Jacal

El objetivo principal de Jacal es lo que actualmente se denomina “animación de arquitecturas”. Esto es, poder visualizar una simulación de cómo se comportaría, en la práctica, un sistema basado en la arquitectura que se ha representado (20).

UML

Lenguaje Unificado de Modelado o UML es el lenguaje de modelado de sistemas de *software* más conocido y utilizado en la actualidad; está respaldado por el OMG[4] (por sus siglas en *inglés Object Management Group*). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como: procesos de negocio, funciones del sistema y aspectos concretos (esquemas de bases de datos y componentes reutilizables). Se aplica a varias metodologías de desarrollo de *software* como es RUP, pero no precisa qué metodología o proceso usar (21).

Ya se ha dicho que UML no es en modo alguno un ADL en el sentido usual de la expresión. Los manuales contemporáneos de UML carecen del concepto de estilo y sus definiciones de arquitectura no guardan relación con lo que ella significa en el campo de los ADLs. Sin embargo, debido al hecho de que constituye una herramienta de uso habitual en el modelado, importantes autoridades en ADLs, han investigado la posibilidad de utilizarlo como metalenguaje.

1.8 Conclusiones

En este capítulo se mostró la importancia de la arquitectura y sus objetivos. Se presentaron los estilos arquitectónicos fundamentales, una breve reseña de los patrones, las ventajas de su uso, así como, una pequeña descripción de las metodologías, lenguajes de modelado y herramientas CASE necesarias para culminar satisfactoriamente el proyecto.

[4] La OMG (*Object Management Group*) es una asociación sin fines de lucro, formada por grandes corporaciones que se encargan de la definición y el mantenimiento de estándares para aplicaciones de la industria de la computación.

Como resultado de este estudio se pudo determinar que el proyecto se desarrollará utilizando la metodología RUP, se modelará con el lenguaje de UML y en *Visual Paradigm*. Todo lo cual se considera las bases para la arquitectura del Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles.

Capítulo 2: Línea base de la arquitectura

2.1 Introducción

En este capítulo se describen las características fundamentales de la arquitectura del Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles: las concepciones generales, el estilo seleccionado, los f*rameworks*, patrones utilizados, así como, la explicación de las herramientas y tecnologías escogidas para el desarrollo del proyecto.

2.2 Objetivos y restricciones Arquitectónicas

La arquitectura de *software* es guiada por los requisitos no funcionales, es por tal motivo que se explican con detalles en los siguientes subepígrafes.

2.2.1 Usos

Mensajes y Textos

Los mensajes de error, deberán ser en idioma inglés y tener una apariencia estándar. Además tendrán que ser lo suficientemente informativos como para dar a conocer la gravedad del error.

2.2.2 Fiabilidad

Tiempo de disponibilidad del sistema

El sistema debe estar disponible como mínimo un 90% de su vida útil. Esto dependerá también de factores externos a nuestro sistema como la interacción con el SMSC (*Short Message Service Center*)[5].

[5] Centro de servicios de mensajes cortos.

Realización de mantenimientos preventivos

No se debe afectar la vitalidad del sistema por mantenimientos. Estos procesos deberán realizarse en tiempo de ejecución.

2.2.3 Seguridad

Permitir canales cifrados de transmisión de datos entre el cliente y el sistema

Deberán asegurarse los canales por donde se intercambien las contraseñas entre los usuarios y el sistema.

Permitir registro de eventos de sistema (LOGs)

El sistema deberá tener la capacidad de registrar una cantidad finita de eventos normales o anormales del sistema, para poder desarrollar una auditoría.

Deberán ser registrados como mínimo los siguientes eventos:

- Operaciones administrativas en el sistema.
- Solicitud de descarga de contenido realizada por un cliente.
- Entrega de un contenido solicitado por un cliente.
- Expiración de una solicitud para descargar un contenido.
- Mantener registros de todos los errores que puedan ocurrir en cualquier momento de la interacción del usuario con el sistema y del mal funcionamiento que este pueda tener en un instante determinado.

El sistema debe proteger la integridad de la información y los contenidos

El sistema debe hacer que la información sea segura y esté disponible en el momento necesario, con el rendimiento adecuado, en el dispositivo correcto y sólo para quien lo precise, y los contenidos no se deben distribuir libremente entre los usuarios finales.

2.2.4 Confiabilidad

El sistema debe ser capaz de recuperarse ante la ocurrencia de fallos, así como, de proteger la información y contenidos.

2.2.5 Eficiencia

Tiempo de respuesta por transacciones

Los tiempos de respuesta serán reducidos al máximo posible, teniendo en cuenta el procesamiento necesario para responder a cada petición.

- Rendimiento: el 95% de las peticiones deben procesarse en menos de 1 minuto. El tiempo que el usuario dedique a la elección de sus contenidos y de sus acciones no se comprenderán dentro de este límite de tiempo.

- Capacidad: La cantidad de peticiones que pueden ser procesadas simultáneamente varía en dependencia de los parámetros de configuración, sujetos al *hardware* con que se cuente.

2.2.6 Documentación del sistema

Se deberá elaborar un manual para los usuarios.

El manual de usuario es un documento técnico que intenta dar asistencia a cualquier persona y debería ser entendido por cualquier usuario principiante y ser de gran utilidad a usuarios avanzados.

Este manual deberá contener aspectos relacionados con el uso del sistema y de todos los objetos generados por este. En él, deben abordarse las definiciones principales del sistema, el trabajo con los archivos que se generan (LOGs y CDRs), los posibles errores que pudieran ocurrir y las soluciones viables.

2.3 Concepciones Generales

2.3.1 Metodología de desarrollo.

El sistema se elaboró utilizando RUP como metodología de desarrollo de *software*. Esta metodología genera gran volumen de documentación, la cual es muy útil en el momento de la entrega al cliente y permite la continuidad del proyecto en equipos de desarrollo variables. Además de esto, es adaptable para proyectos a largo plazo y establece refinamientos sucesivos de la arquitectura del sistema. RUP constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas informáticos (11).

Otro punto de peso en esta decisión, es la experiencia acumulada por el equipo de proyecto en el uso de dicha metodología, que evita la pérdida de tiempo en períodos de adiestramiento.

La utilización de dicha metodología se refuerza utilizando practicas provenientes de otras como es el caso del Desarrollo guiado por Pruebas (TDD, del inglés Test Driven Development) que plantea el desarrollo de pruebas de forma paralela al de las funcionalidades del sistema. Esta práctica persigue como objetivo principal el permitir detectar los errores tempranamente, incrementando finalmente la calidad del producto desde el punto de vista de la libertad de errores.

2.3.2 Especificaciones

Las especificaciones establecen, de forma clara, todas las características, los materiales y los servicios necesarios para producir componentes destinados a la obtención de productos. Estos incluyen requerimientos para su conservación, su empaquetamiento, almacenaje y marcado; así como, los procedimientos para determinar su éxito y medir su calidad.

La utilización de especificaciones para la creación de un producto permite cambiar en cualquier momento la implementación de la misma realizando pocos o ningún cambio en la aplicación.

JPA

Java Persistence API (JPA) es un estándar para la persistencia de datos en aplicaciones Java, de forma que se simplifique el desarrollo de la persistencia de datos.

Es una API de persistencia de POJOs (*Plain Old Java Object*), o sea, los objetos simples que no heredan ni implementan otras clases. Permite que el mapeo objeto-relacional se realice mediante anotaciones en las propias clases de entidad, y que no se requieran ficheros descriptores XML (22).

SAMS-M

SAMS-M (del inglés *Server API for Mobile Services*) es una especificación desarrollada por Nokia y Sun Microsystems en el 2004.

Plantea una solución estándar para el envío y recepción de mensajes, independientemente de las características de los operadores. Mantiene una arquitectura basada en la utilización de *drivers* específicos para cada protocolo. Estos *drivers* liberan al desarrollador de dichos detalles, que incrementa, de esta forma, su productividad (23).

Estándar de Codificación

Al comenzar un proyecto de *software*, es necesario establecer un estándar de codificación para asegurarse de que todos los desarrolladores e integrantes del proyecto trabajen de forma coordinada. El código fuente del sistema debe reflejar un estilo armónico, como si una misma persona lo hubiera escrito. Un estándar de codificación comprende todos los aspectos de la generación de código.

Cuando el proyecto de *software* incorpore un código fuente ya utilizado, o bien cuando se realice el mantenimiento; el estándar de codificación debe establecer cómo operar con ese código. El Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles se desarrolló bajo las pautas de codificación del Polo de Telemática y Telecomunicaciones versión 1.0 (24).

2.4 Estilo Arquitectónico Seleccionado

Para la realización del núcleo de la plataforma se decidió utilizar el estilo de arquitectura basada en componentes, la cual se centra en el diseño y construcción de sistemas computacionales que utilizan componentes de *software* reutilizables.

En esencia, un componente es una pieza de código preelaborado que encapsula alguna funcionalidad expuesta a través de interfaces estándar.

El uso de este estilo permite alcanzar un mayor nivel de reutilización de *software*, que las pruebas sean ejecutadas por componentes antes de evaluar el conjunto completo, actualizar o agregar componentes sin afectar otras partes del sistema y ciclos de desarrollo más cortos. Es la forma que posibilita emplear códigos y conocimientos ya existentes (25, 26).

2.5 *Frameworks* para el desarrollo

Los *frameworks* se pueden considerar como una aplicación genérica y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta. Los objetivos principales que persiguen son: acelerar el proceso de implementación, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones (27).

Fueron diseñados para facilitar el desarrollo de *software*, permitiendo a los programadores dedicar más tiempo a la implementación de los procesos referentes al sistema y abstraerse de otras tareas, necesarias, pero que no son el centro del desarrollo tales

como: acceso a datos, manejo de transacciones, etc. Contienen un número determinado de componentes implementados, con sus interfaces bien puntualizadas que pueden ser reutilizados o redefinidos.

A partir de la organización de la arquitectura, se propone la utilización de los siguientes *frameworks*:

Spring Framework 2.5

Spring está compuesto por un conjunto de módulos, de los cuales se pueden tomar aquellos que faciliten la implementación del proyecto en cuestión. Ideado principalmente por Rod Johnson[6], es libre y de código abierto desde febrero de 2003.

El centro de Spring está basado en el principio de Inyección de Dependencias. Esta técnica hace externa la creación y el manejo de las dependencias de las clases, con lo que se logra una mayor limpieza y claridad en el código (28).

Este *framework* brinda soporte además para la utilización de la Programación Orientada a Aspectos (*AOP: Aspect Oriented Programming*). La cual permite aislar los conceptos que se cruzan en la programación como el manejo de transacciones, manejo de trazas y seguridad en aspectos que luego se le aplican a las clases (29).

Spring mantiene una facilidad de integración con otras tecnologías, como por ejemplo, Java Persistence API (JPA). Además es un *framework* maduro y con una amplia comunidad de desarrollo.

JUnit 4.0

Es un *framework* desarrollado para realizar pruebas automatizadas a los sistemas informáticos durante la etapa de construcción. Su objetivo principal es evaluar el funcionamiento de cada uno de los métodos dentro de las clases que conforman el *software* (30).

Este framework facilita la aplicación de la practica TDD (del inglés *Test Driven Development*). Incluye formas de ver los resultados que pueden ser en modo texto o gráfico. Es uno de los más utilizados de su tipo para el lenguaje Java, cuenta con una amplia comunidad de desarrollo y es bastante maduro.

[6] Rod Jonson es una de las autoridades del mundo sobre el desarrollo de Java y JEE. Es un gran autor, consultor experimentado, y desarrollador de código abierto, así como un popular conferencista.

Hibernate 3.3

Hibernate es un *framework* potente y de alto rendimiento que se especializa en la persistencia de datos, también se define como una herramienta de mapeo objeto-relacional para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.

Dicho *framework es software* libre; permite desarrollar clases persistentes orientadas a objetos, incluyendo la asociación, la herencia, el polimorfismo, la composición y las colecciones y además tiene una implementación de la especificación JPA (31).

2.6 Patrones Utilizados

El establecimiento de los patrones es lo que posibilita el aprovechamiento de la experiencia acumulada en el diseño de aplicaciones (7). Algunos de los más significativos en el Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles son:

Facade

El patrón fachada tiene como objetivo facilitar el uso de un subsistema, estableciendo una interfaz simple y de fácil comprensión para el cliente. En esta interfaz se añaden métodos que representan las funcionalidades más utilizadas, y se habilita un punto de entrada unificado al subsistema en cuestión (7).

Otra de las facilidades que brinda este patrón es el resumen de las llamadas a varios métodos en uno solo en sistemas donde se hagan invocaciones a través de la red, lo cual mejora considerablemente el rendimiento del sistema (7).

Este patrón se utiliza en todos los componentes de la plataforma. Cada uno suministra una interfaz simple para la interacción desde otros, lo cual resulta una ventaja.

Factory Method

El objetivo de este patrón es establecer una interfaz única para la creación de objetos bajo un criterio determinado (7).

Este patrón se utiliza en el componente *Transcoder* con el objetivo de seleccionar qué implementación de la clase encargada de hacer los procesos de conversión sobre los contenidos, se debe utilizar.

Template Method

El objetivo de este patrón es establecer el esqueleto de un algoritmo que facilite a las subclases definir el comportamiento del algoritmo, en dependencia de las necesidades de cada una. Esto permite establecer una interfaz común, y dar implementaciones concretas en las subclases que lo requieran(7).

El patrón template method se utiliza en el componente *Content Delivery* con el objetivo de establecer una interfaz común para los diferentes mecanismos de entrega de contenidos, al hacer posible que cada uno realice la implementación específica en dependencia de cómo funcione.

Singleton

El patrón singleton se utiliza cuando se desea tener solo una instancia de una clase y obtener, en todo momento, dicha instancia (7).

Este patrón se utiliza prácticamente en toda la plataforma. Esta función se logra utilizando el *framework* Spring, quien tiene una de sus implementaciones, liberando así a los programadores de esta responsabilidad.

Adapter

El patrón adapter tiene como objetivo adecuar un componente de *software* a las necesidades de un usuario determinado. Como generalidad este patrón se utiliza cuando se desea utilizar un componente desarrollado independientemente del proyecto y por tanto las interfaces del mismo no se ajustan a las necesidades del nuevo sistema (7).

Este patrón se utiliza en el componente *transcoder* con el objetivo de adaptar las interfaces del Alembik a las necesidades de la plataforma.

2.7 Entorno de desarrollo

2.7.1 Lenguaje de Programación

Java 6.0

Java es un lenguaje de programación que ofrece la potencia del diseño orientado a objetos con una sintaxis fácilmente accesible y un entorno robusto y agradable.

Con Java se realiza un descubrimiento de la mayor parte de los errores durante el tiempo de compilación; así, lo que es rigidez y falta de flexibilidad se convierte en eficacia. Este lenguaje posee una gestión avanzada de memoria llamada gestión de basura (*Garbage Collector*), y un manejo de excepciones orientado a objetos integrados.

Es un lenguaje maduro, de fácil aprendizaje, interactivo, uno de los más utilizados para el desarrollo de aplicaciones para celulares y cuenta con un conjunto de herramientas libres que facilitan considerablemente las tareas (36).

2.7.2 Plataforma

Java EE 5.0

Java Enterprise Edition o Java EE, es una plataforma de programación para desarrollar y ejecutar *software* de aplicaciones en lenguaje de programación Java con arquitectura de N niveles distribuidos. Se basa en componentes de *software* modulares y se ejecuta sobre un servidor de aplicaciones (38).

Se utiliza Java EE porque es una plataforma libre, robusta, madura, segura; mayormente se emplea para crear *software* empresarial utilizando el lenguaje Java y cuenta con un buen soporte.

2.7.3 Entornos de Desarrollo Integrado

Eclipse Ganymede

Eclipse es un entorno de desarrollo integrado de código abierto, que se basa en la Plataforma de cliente enriquecido (del Inglés *Rich Client Platform* RCP) (39).

Eclipse Ganymede es una nueva versión del conjunto de herramientas de desarrollo de *software* del proyecto Eclipse. Posibilita desarrollar aplicaciones web y *Web Services* con diferentes servidores de aplicaciones (40).

Esta aplicación se utiliza porque es multiplataforma, libre, robusta, madura y es uno de los entornos de desarrollo más utilizado a nivel mundial para programar en Java. Es fácil de utilizar y proporciona el aprendizaje de los nuevos componentes que incorpora.

2.7.4 APIs

Alembik 1.0

Alembik es una aplicación Java para proporcionar servicios de transformación u optimización de diferentes tipos de medios (imagen, audio, vídeo, etc.). La arquitectura de Alembik está basada y es totalmente compatible con OMA STI (por sus siglas en inglés *Open Mobile Alliance Standard Transcoding Interface*).

Esta aplicación se utiliza porque es libre, multiplataforma, crea y modifica los contenidos para los celulares como le hace falta a la plataforma para entregarlos adecuadamente; tiene un buen rendimiento y es una de las mejores alternativas de su tipo. Además, permite especificar tamaño, dimensión, duración y formato del archivo deseado, y es relativamente fácil de usar. (41)

2.7.5 Servidor de Base de Datos

PostgresSQL 8.3.7

Es un servidor de base de datos objeto-relacional de *software* libre, publicado bajo la licencia BSD. El código fuente de Postgres está disponible en Internet. También es el gestor más potente entre las alternativas libres. (42).

Posibilita alta concurrencia, lo que permite a un proceso escribir y a otros acceder a la misma tabla sin necesidad de bloqueos. Esta estrategia es superior al bloqueo por tablas o por filas, común en otros gestores (MySQL por ejemplo), en tanto se elimina la necesidad del uso de bloqueos explícitos; tiene soporte nativo para gran variedad de tipos de datos y los usuarios pueden crear sus propios tipos; permite la herencia entre tablas, lo que facilita el desarrollo orientado a objetos; tolera varios lenguajes (C, C++, pl/Perl, plPHP, pl/Phyton, Java PL, pl/sh, pl/Tcl); lo que admite utilizar la potencia de dichos lenguajes, desde bifurcaciones y bucles, hasta programación funcional dentro del gestor; soporta vistas, *triggers*, cursores, claves ajenas, consultas anidadas o *subselect*.

Esta herramienta se emplea porque es libre, rápida, segura, multiplataforma y gracias a su licencia BSD, no se prohíbe la utilización del código para ser comercializado y acepta la replicación. Con ella se pueden almacenar los diferentes contenidos de la plataforma en una base de datos y acceder a estos en cualquier momento.

2.7.6 Servidor Web

Apache Tomcat 6.0

Apache Tomcat es un servidor *web* que implementación de las tecnologías *Java Servlet* y *Java Server Pages*, por esta razón, funciona en cualquier sistema operativo que disponga de la máquina virtual Java. Se desarrolla en un entorno abierto y participativo, publicado bajo la licencia del *software* de Apache. Se usa fundamentalmente en entornos con alto nivel de tráfico y alta disponibilidad (43).

Esta herramienta es utilizada porque es libre, gratis, robusta y fácil de instalar. Tomcat ocupa muy poco espacio y es muy fiable. Es el servidor *web* más utilizado, el más potente para el lenguaje Java y es compatible de forma integrada con muchas aplicaciones.

2.8 Gestión de Proyectos

DotProject 2.1

Para la gestión de proyectos se utilizó la herramienta Dotproject, que brinda la asignación de recursos, la gestión de documentación, y ayuda así a organizar un proyecto en diferentes tareas y en un tiempo determinado. DotProject está construido por aplicaciones de código abierto y es mantenida por un pequeño grupo de voluntarios. Es una aplicación basada en WEB, multiusuario, soporta varios lenguajes, es software libre y el personal tiene experiencia en su utilización (33).

Trac 0.10.3

Trac está optimizado para *Subversion*. Provee una plataforma completa para la colaboración entre usuarios y desarrolladores, interfaz de usuario amigable, crea una entrada en la wiki a cada cambio en el repositorio; seguimiento y documentación de errores, tareas y cambios; integrado con Eclipse mediante Subeclipse y Mylyn, y permite asignar errores (archivos) a la tarea correspondiente, además de notificar al usuario de dichas tareas (34).

Esta es una herramienta muy utilizada, una de las mejores alternativas libres de su tipo, estable y con una gran comunidad de desarrollo.

2.9 Gestión de Configuración

2.9.1 Sistema operativo

Debian 4.0

Para el desarrollo de la plataforma se usó el sistema operativo Debian, que es libre y que utiliza como núcleo Linux, pero la mayor parte de las herramientas básicas vienen del Proyecto GNU.

Debian viene con *softwares* precompilados y empaquetados en un formato amigable y para una instalación sencilla, todos ellos de forma gratuita (32).

Se escogió este sistema operativo ya que es libre, maduro, estable, fácil de utilizar, existe una amplia comunidad de desarrollo y mantiene unos bajos niveles de consumo de recursos.

La distribución de Debian va orientada principalmente a servidores y a máquinas clientes para desarrolladores.

2.9.2 Sistema de Integración continua

PBS 2.0

La integración continua (*continuos integration*) es una práctica, que consiste en hacer integraciones automáticas de un proyecto con la mayor frecuencia posible para detectar fallos cuanto antes. Entendemos por integración, la compilación y ejecución de *test* de todo un proyecto (35).

Cada cierto tiempo la herramienta descarga las fuentes desde el gestor de versiones, lo compila, ejecuta los *test* y genera los informes.

Esta aplicación se utiliza porque cumple cabalmente con las necesidades de integración. Es un sistema desarrollado dentro de la universidad, no hay que pagarlo, está probado, tiene tiempo en explotación y el personal tiene conocimiento sobre su utilización.

2.9.3 Control de Versiones

***Subversion* 1.4.5**

Es un *software* de sistema de control de versiones. Es libre bajo una licencia de tipo Apache/BSD. Una característica importante de *Subversion* es que, los archivos

versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común en cierto punto del tiempo (36).

Subversion ofrece las funciones necesarias para el control de versiones de un conjunto de archivos. Es maduro, estable, existe una amplia comunidad de desarrolladores y es ampliamente utilizado a nivel mundial.

2.9.4 Cliente del control de Versiones

RapidSVN

Es un cliente gratuito de código abierto para el sistema de control de versiones *Subversion*. Se encarga de manejar ficheros y directorios a lo largo del tiempo. Los ficheros se almacenan en un repositorio central que funciona como un servidor de ficheros ordinario, salvo que recuerda todos los cambios que se hayan hecho a sus archivos y directorios. Esto permite que pueda recuperar versiones antiguas y examinar la historia de cuándo, quién y cómo cambió sus datos.

RapidSVN es muy simple de usar, y permite hacer tanto las operaciones básicas como bajar y subir el código fuente, hasta las más complejas como creaciones de ramas y fusión (37).

2.10 Conclusiones

En este capítulo se describieron las restricciones arquitectónicas, las características fundamentales de la arquitectura del Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles: el estilo seleccionado, los patrones arquitectónicamente significativos, las concepciones generales, los *frameworks* utilizados y las principales tecnologías y herramientas de desarrollo de *software*.

Capítulo 3: Definición de la arquitectura

3.1 Introducción

En esta sección se describe la arquitectura del Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles y cada uno de sus componentes; se explica el flujo de procesos y además se muestra un conjunto de vistas arquitectónicas representadas con UML.

3.2 Características fundamentales del sistema

La solución propuesta se divide en tres subsistemas:

- El núcleo de la plataforma. Se basa en recibir la orden de descarga del contenido, optimizarlo para el dispositivo que lo pide, enviárselo y cobrar. Este subsistema presenta una arquitectura orientada a componentes y brinda servicios que permiten administrar todo el contenido que se maneja.
- El portal WAP. Es el enlace para la descarga, además brinda servicios de noticias (nacionales, internacionales, deportivas, culturales y meteorológicas). Se desarrolló utilizando el patrón Modelo-Vista-Controlador.
- El portal *web*. Se desarrolló exclusivamente con fines administrativos. Su única función es la de gestionar el contenido, los proveedores, los servicios y algunas configuraciones de la plataforma y el portal WAP. También se desarrolló utilizando el patrón Modelo-Vista-Controlador.

La comunicación entre estos subsistemas se hace a través de WEB_Services; los que además presentan seguridad, tanto a nivel del canal de trasmisión (https) como a nivel de Mensajes[7].

[7] La Seguridad a nivel de Mensajes se garantiza usando "ws-security", que es un estándar de seguridad para los servicios *web*.

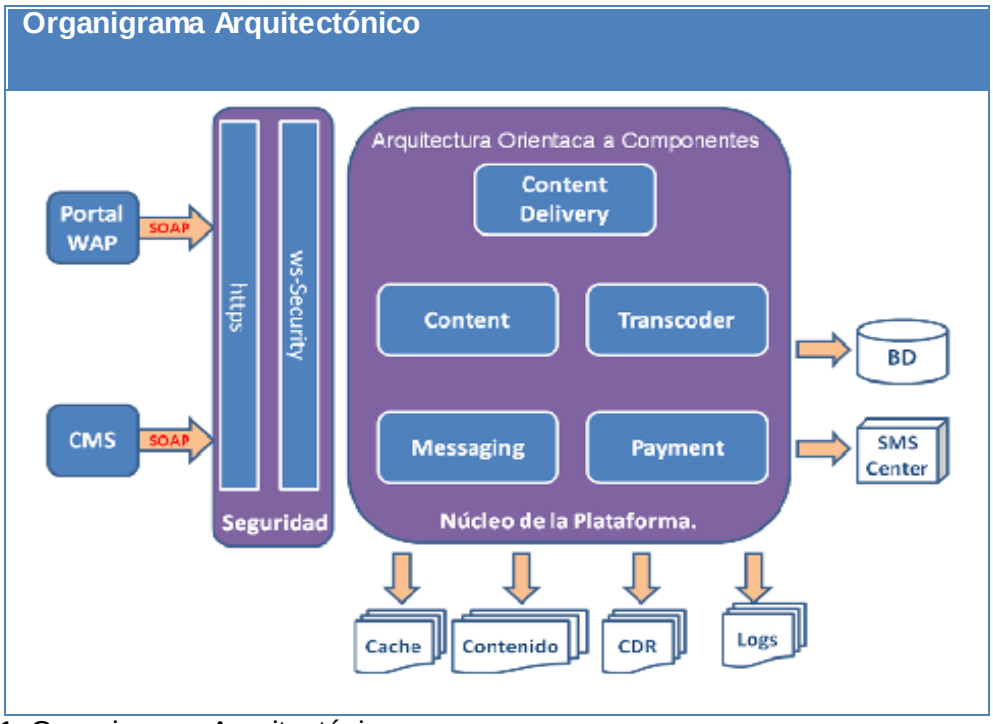


Figura 1: Organigrama Arquitectónico.

Este trabajo se basa en el Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles, que es la parte genérica y comercializable. Se mencionan los portales WAP y *web* porque son parte de la solución que se le entregará al cliente Cubacel, pero perfectamente pueden cambiarse o quitarse.

3.3 Descripción de los Componentes

Content Delivery: Brinda los mecanismos para registrar las nuevas peticiones. Organiza todo el proceso de venta del contenido (optimización, entrega y cobro). Este componente tiene integrados varios mecanismos de descarga y selecciona el más apropiado, teniendo en cuenta las particularidades del teléfono, además, permite la adición de nuevos métodos de descarga.

Content: Brinda los mecanismos para la administración de los contenidos desde otra aplicación (CMS), y es el encargado de gestionarlos.

Transcoder: Se especializa en la optimización de los contenidos para un tipo de dispositivo determinado.

Payment: Brinda los mecanismos para realizar el cobro de una solicitud. Tiene integrado un método basado en la generación de los CDR (por sus siglas en inglés *Call Detail Record* o Registro Detallado de Llamadas), y permite la adición de otros nuevos.

Messaging: Este componente utiliza una implementación de la especificación SAMS-M anteriormente descrita y es el encargado de enviar los mensajes de texto a los usuarios, indicando la URL donde pueden descargar el contenido solicitado.

Todos estos componentes permiten realizar modificaciones internas sin alterar el sistema en sí.

3.4 Flujo de procesos en la Plataforma

El usuario accede al portal WAP, dicho portal solicita y obtiene el listado de contenidos del componente “Content”, cuando el usuario selecciona el contenido a descargar, el portal WAP informa al componente “Content Delivery”.

El módulo “Content Delivery” recibe la orden, solicita al módulo “Content” que la registre en la base de datos, crea la URL del descriptor y ordena enviarla a través del componente “Messaging”.

El Cliente accede a la URL del descriptor y “Content Delivery” obtiene los datos de “Content”, crea el descriptor y comanda a “Messaging” para que lo envíe al usuario. En caso de que el portador del dispositivo móvil acepte descargar el contenido tras leer el descriptor, el componente “Content Delivery” solicita el contenido al “Content.”

El módulo “Content” busca en la “Cache” si existe el contenido que se solicita, optimizado para el “*user agent*” especificado; en caso de que no esté, lo manda a construir al “Transcoder” el que, al terminar, lo almacena en dicha “Cache” e informa al “Content Delivery”.

El componente “Content Delivery”, obtiene del “Content” el archivo especificado y optimizado en la caché y se lo envía al usuario.

El descriptor que anteriormente había recibido el cliente tiene un campo que informa el resultado de la recepción del archivo al terminar su envío. En caso de éxito el “Content Delivery” manda al “Content” a borrar la orden de la base de datos y ordena el cobro al componente “Payment”; este último genera un archivo y lo guarda.

Si el descriptor informara algún problema se tomarían medidas en dependencia de la anomalía presentada, por ejemplo, el reenvío del mensaje al usuario si se ha interrumpido la conexión.

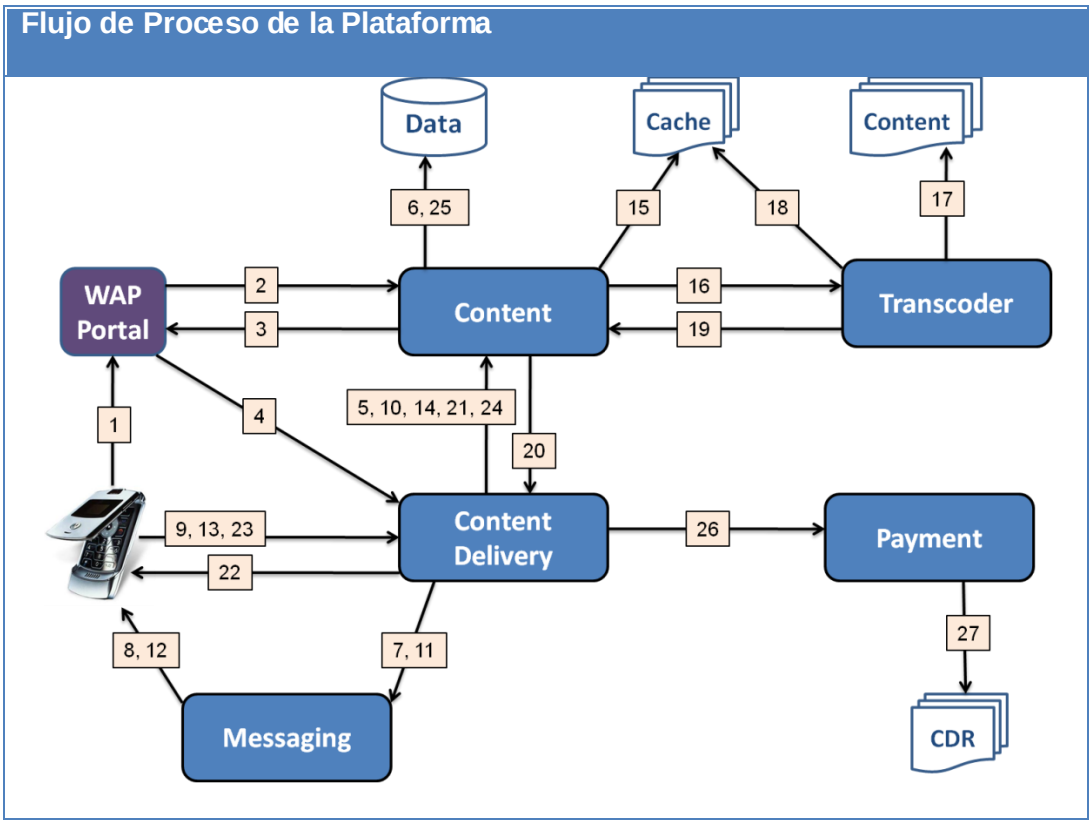


Figura 2: Flujo de Proceso de la Plataforma.

Leyenda:

1. Accede al WAP
2. Solicita el listado de los contenidos
3. Entrega el listado
4. Informa del contenido seleccionado
5. Crea y manda a guardar la orden
6. Guarda la orden
7. Crea la URL del descriptor y ordena enviarla
8. Envía la URL del descriptor
9. Accede a la URL del descriptor
10. Obtiene los datos y crea el descriptor
11. Ordena enviar el descriptor
12. Envía el descriptor

13. Accede a descargar el contenido
14. Solicita el contenido
15. Busca el contenido especificado para ese *User-Agent* (Suponiendo que no está)
16. Solicita optimización del contenido
17. Obtiene el contenido
18. Transforma el contenido y lo guarda
19. Notifica el fin de la transformación
20. Envía el identificador del contenido en la caché
21. Obtiene el archivo con ese identificador
22. Entrega el contenido
23. Notifica el tipo de recepción del archivo (caso de éxito)
24. Manda a borrar la orden
25. Borra la orden
26. Manda a Cobrar
27. Genera los LOGs

3.5 Vista de Casos de Uso

La vista de casos de uso es una de las más importantes, pues brinda la información de de las funcionalidades imprescindibles para el sistema y es la más comprensible para el cliente.

En esta vista se muestra la percepción que tiene el usuario de las funcionalidades del sistema mediante la representación de los actores y casos de uso arquitectónicamente significativos, que son aquellos que sirven para validar la arquitectura propuesta y describen las funcionalidades indispensables para el sistema.

El siguiente diagrama muestra la vista general de casos de uso.

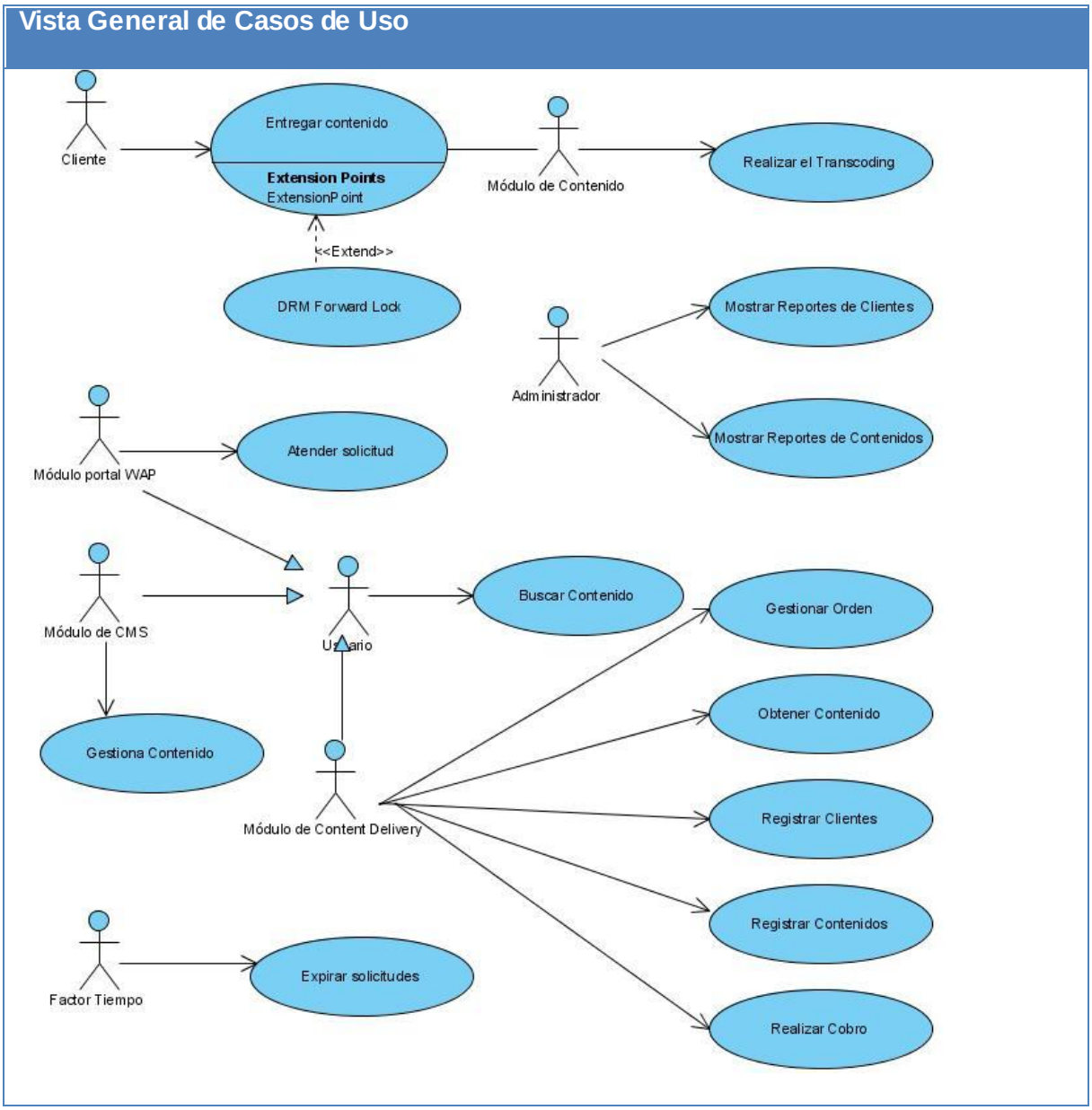


Figura 3: Diagrama General de Casos de Uso.

A continuación se muestran los diagramas por componentes, de dicha vista y se explican brevemente cada uno de los casos de uso.

Componente *Content Delivery*: Organiza todo el proceso de venta del contenido.

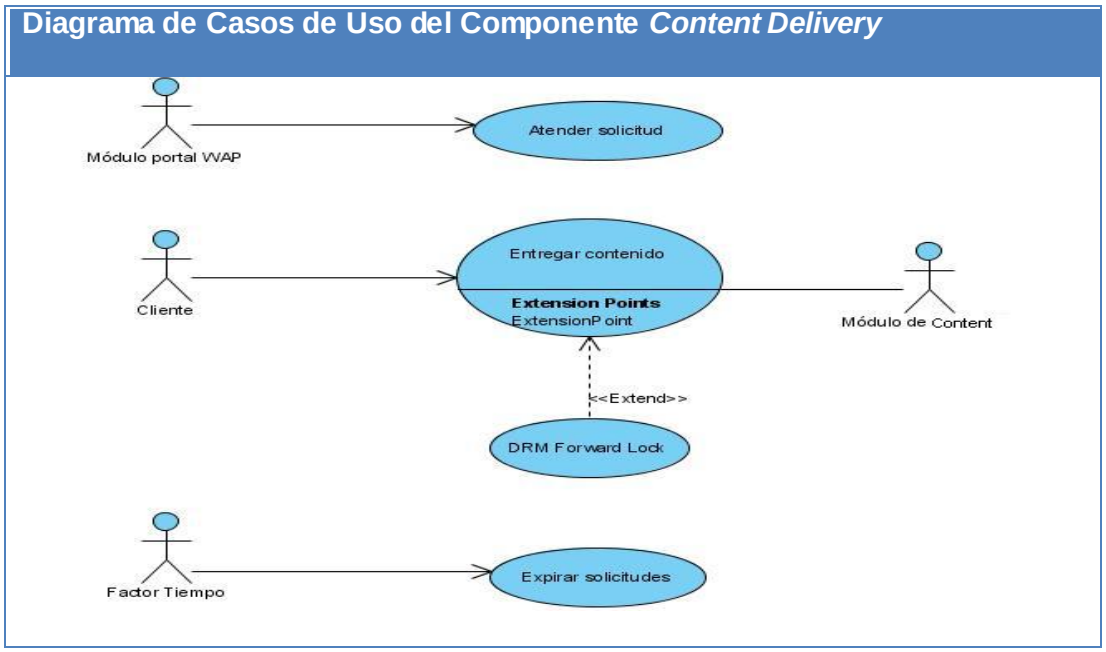


Figura 4: Diagrama de Casos de Uso del Componente *Content Delivery(44)*

CU Atender Solicitud: Atiende la solicitud del contenido que el cliente desea. Se encarga de gestionar que se registren los datos de la solicitud, y enviar al cliente la URL donde debe descargar el descriptor.

CU Entregar Contenido: Entrega el contenido al cliente, se procede al cobro y se eliminan los datos referentes a la orden.

CU Expirar Solicitudes: Si el tiempo de espera de las solicitudes supera el establecido para descargar el contenido, dichas solicitudes cambiarán su estado a expirado y se eliminarán.

CU DRM Forward Lock: Es el encargado de proteger el contenido especificado aplicando DRM con el método de entrega Forward Lock.

Componente *Content*: Es el encargado de gestionar todos los contenidos.

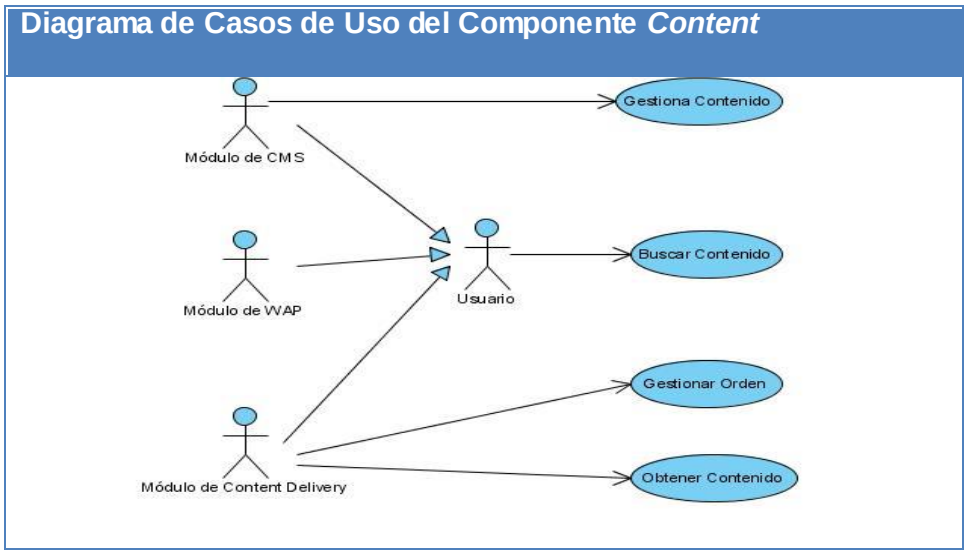


Figura 5: Diagrama de Casos de Uso del Componente *Content (45)*

CU Gestionar Contenido: Permite al Módulo de CMS gestionar los contenidos.

CU Buscar Contenido: Permite al portal Wap y al CMS buscar uno o varios contenidos según el parámetro solicitado.

CU Gestionar Orden: Permite al Módulo Content Delivery gestionar las órdenes de descarga.

CU Obtener Contenido: Permite al Módulo Content Delivery obtener un contenido desde la base de datos.

Componente *Transcoder:* Optimiza los contenidos para un tipo de dispositivo determinado.

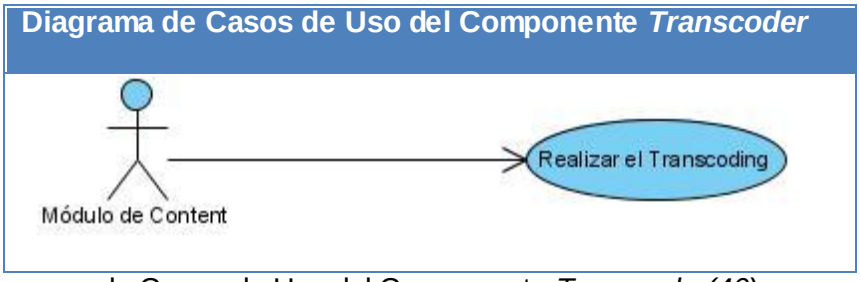


Figura 6: Diagrama de Casos de Uso del Componente *Transcoder(46)*

CU Realizar el Transcoding: Se ocupa de optimizar los contenidos para dispositivos específicos.

Componente *Payment*: Brinda los mecanismos para realizar el cobro de una solicitud.

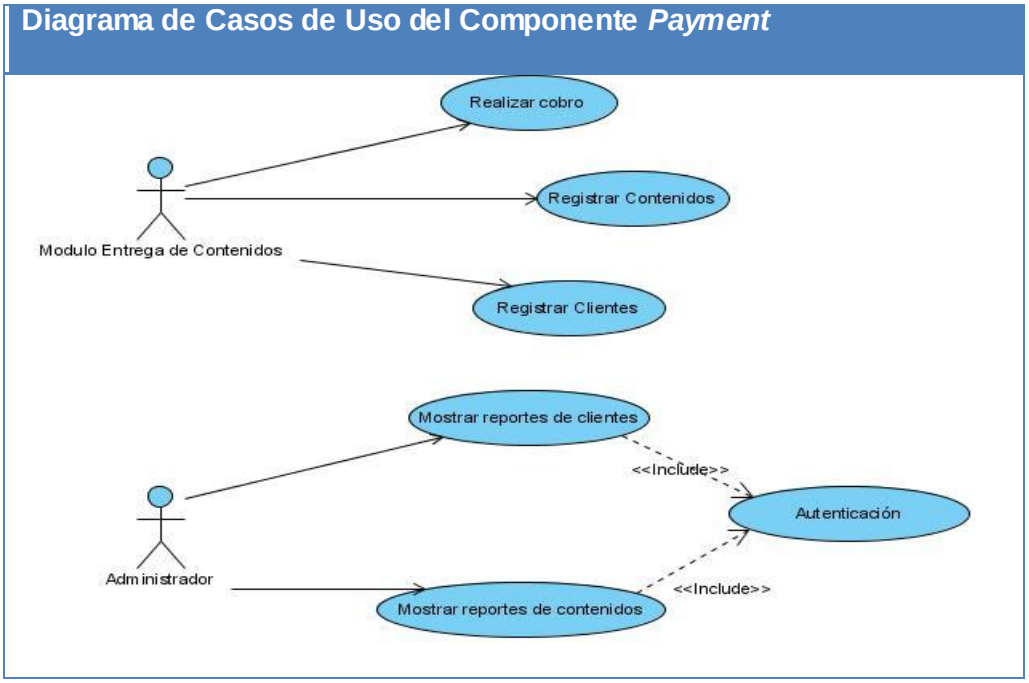


Figura 7: Diagrama de Casos de Uso del Componente *Payment(47)*

CU Realizar Cobro: Guarda todos los datos de la descarga y, a partir de estos, se realiza el cobro.

CU Registrar Cliente: Archiva y registra todos los datos de los clientes.

CU Registrar Contenido: Archiva y registra todos los datos de los contenidos.

CU Mostrar Reporte de Contenidos: Muestra la petición del usuario y los contenidos más descargados. Requiere de la autenticación.

CU Mostrar Reporte de Clientes: Presenta la petición del usuario y los clientes que más contenidos descargan. Requiere de la autenticación.

CU Autenticar: Verifica que el usuario que está accediendo a los datos tenga los permisos requeridos.

3.6 Vista Lógica.

En la descripción de la arquitectura, la vista lógica describe las clases más importantes que formarán parte del ciclo de desarrollo. Se describen los paquetes del sistema y las relaciones que entre ellos existen.

La siguiente figura ilustra la división en módulos del núcleo de la plataforma a construir. Esto se hace con el objetivo de hacer el sistema más reusable, facilitar el mantenimiento y permitir que el equipo de proyecto pueda ir desarrollando los componentes paralelamente.

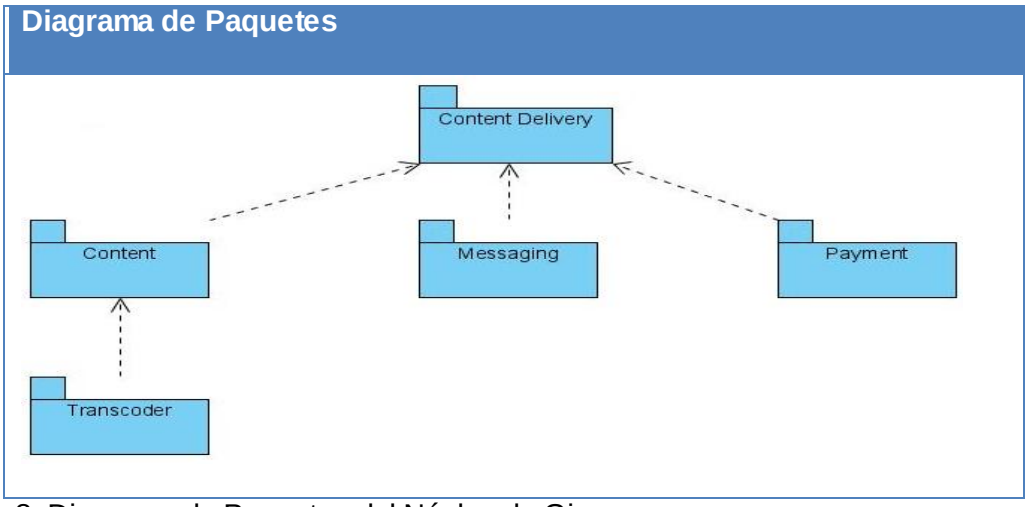


Figura 8: Diagrama de Paquetes del Núcleo de Gina

A continuación se muestran los paquetes arquitectónicamente significativos y los diagrama de clases del diseño de cada uno.

Componente *Content Delivery*

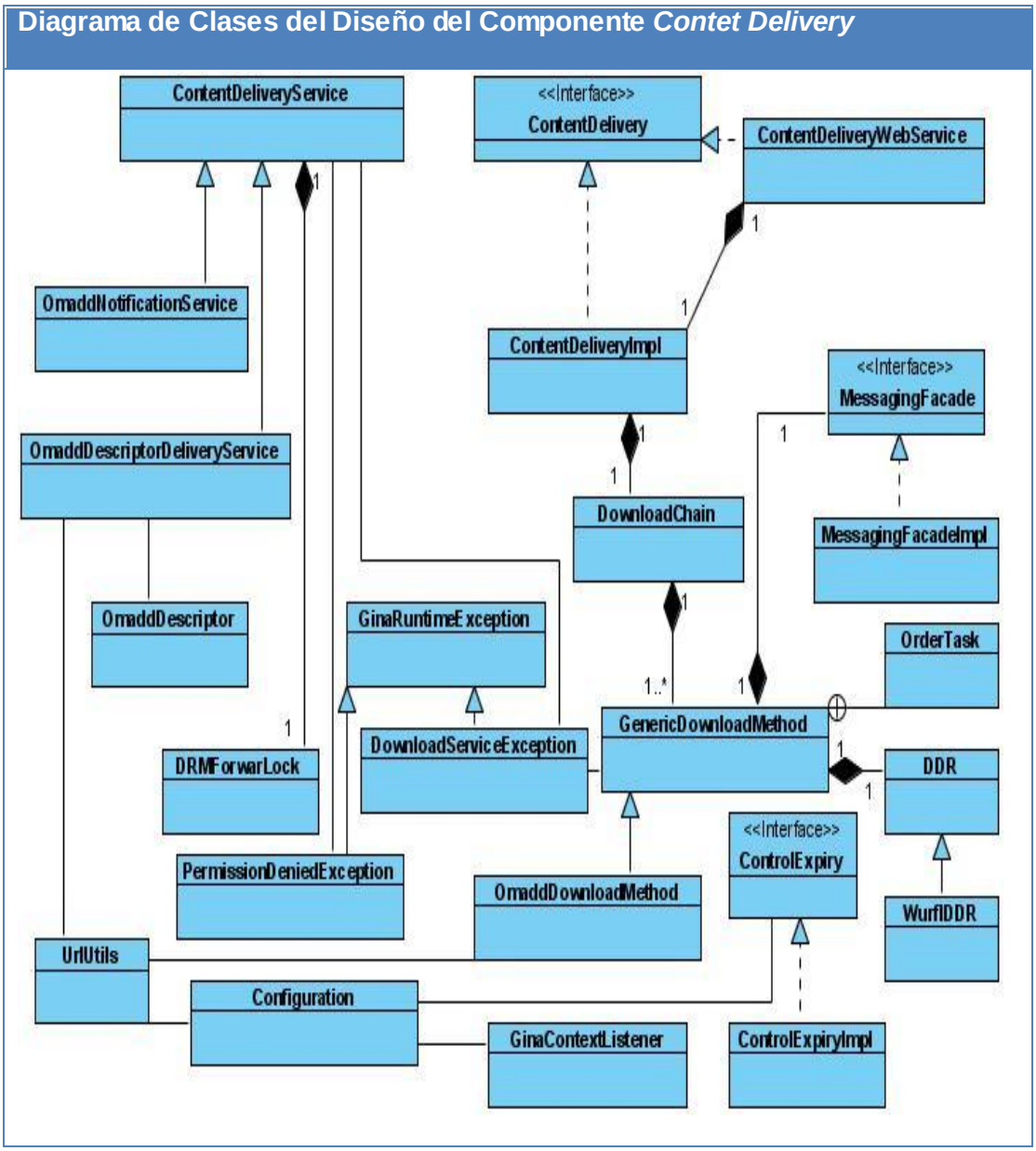


Figura 9: Diagrama de Clases del Diseño del Componente *Content Delivery (44)*

Componente *Content*

Este componente se dividió en cuatro paquetes para hacerlo más comprensible. A continuación, se muestra su estructuración y el contenido de cada uno.

Diagrama de Paquetes de las Clases del Diseño del Componente *Content*

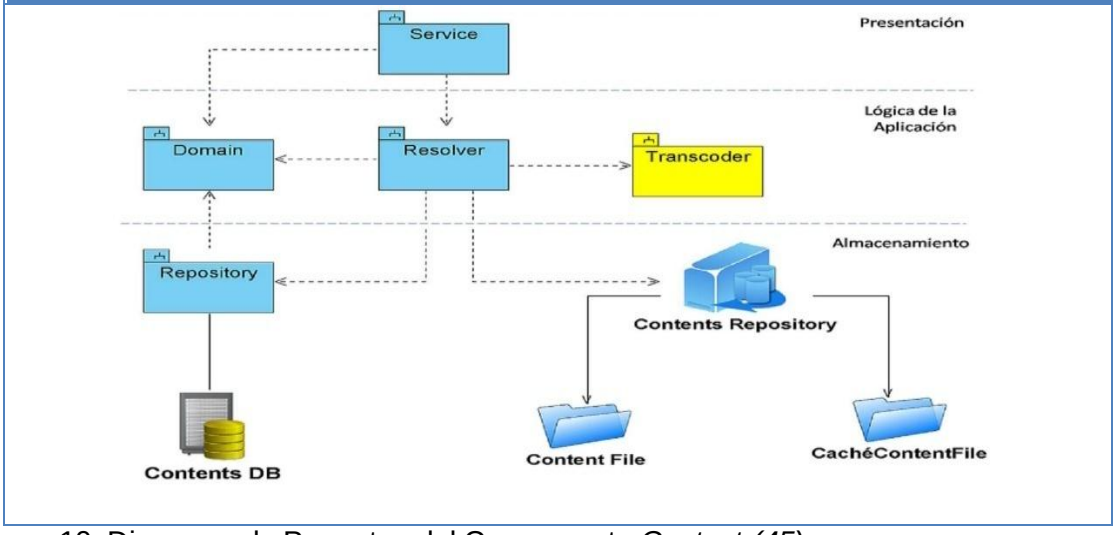


Figura 10: Diagrama de Paquetes del Componente *Content (45)*

Diagrama de Clases del diseño del Componente *Content. Paquete Service*

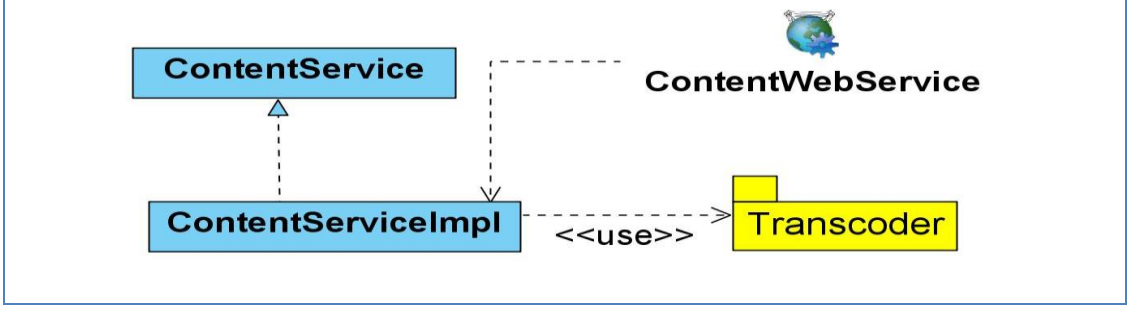


Figura 11: Diagrama de Clases del Diseño del Componente *Content. Paquete Service (45)*

Diagrama de Clases del diseño del Componente *Content. Paquete Domain*

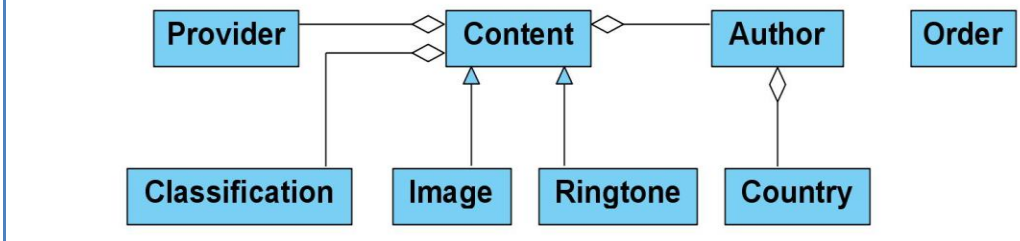


Figura 12: Diagrama de Clases del Diseño del Componente *Content. Paquete Domain (45)*

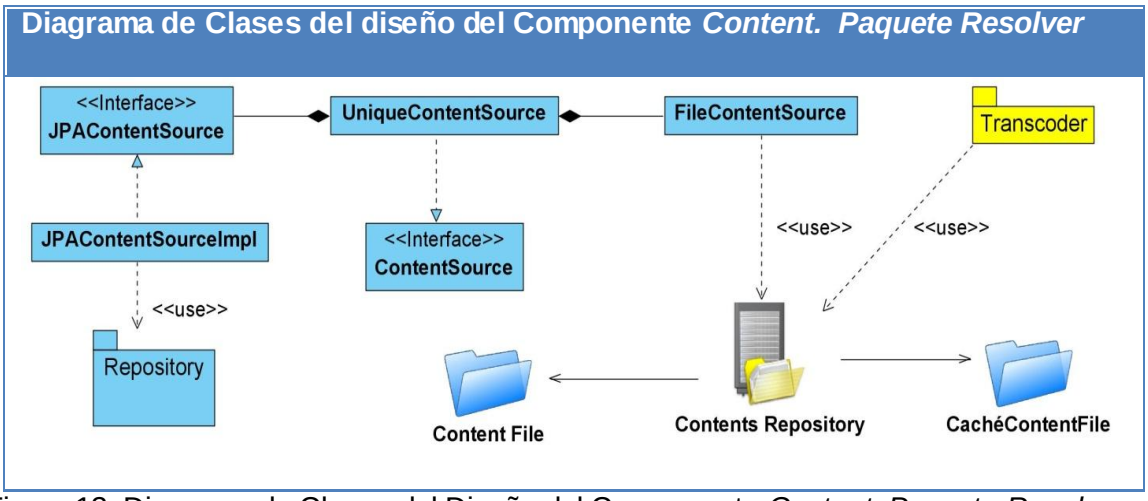


Figura 13: Diagrama de Clases del Diseño del Componente *Content. Paquete Resolver (45)*

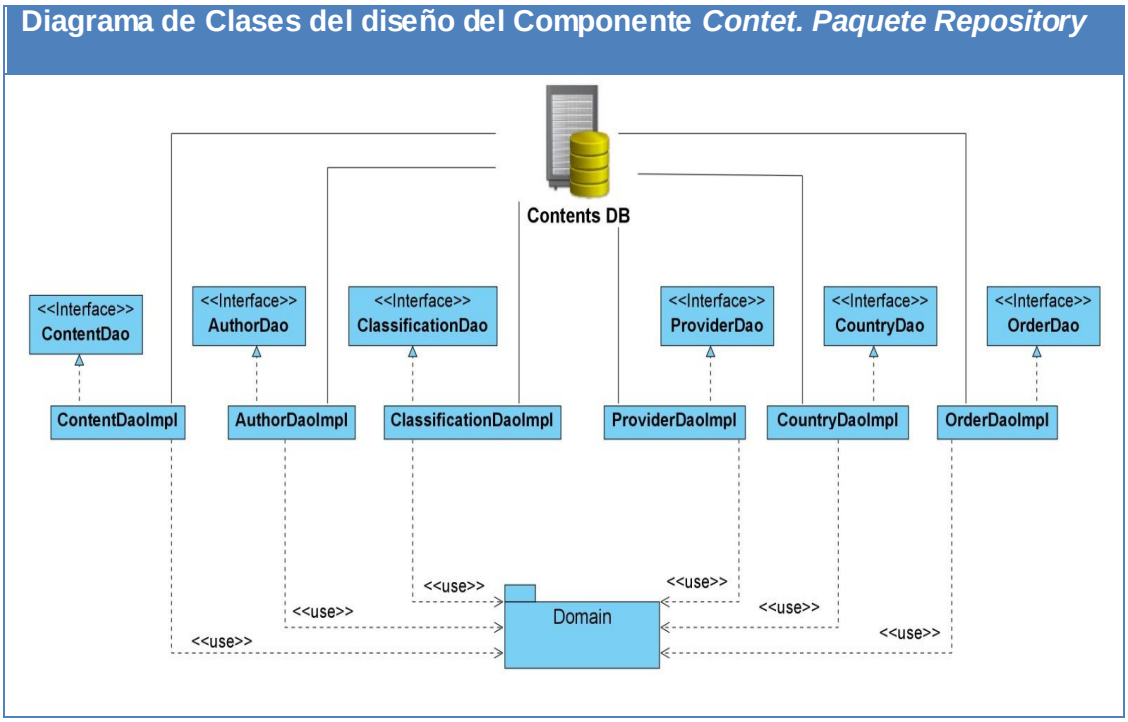


Figura 14: Diagrama de Clases del Diseño del Componente *Content. Paquete Repository (45)*

Componente *Transcoder*

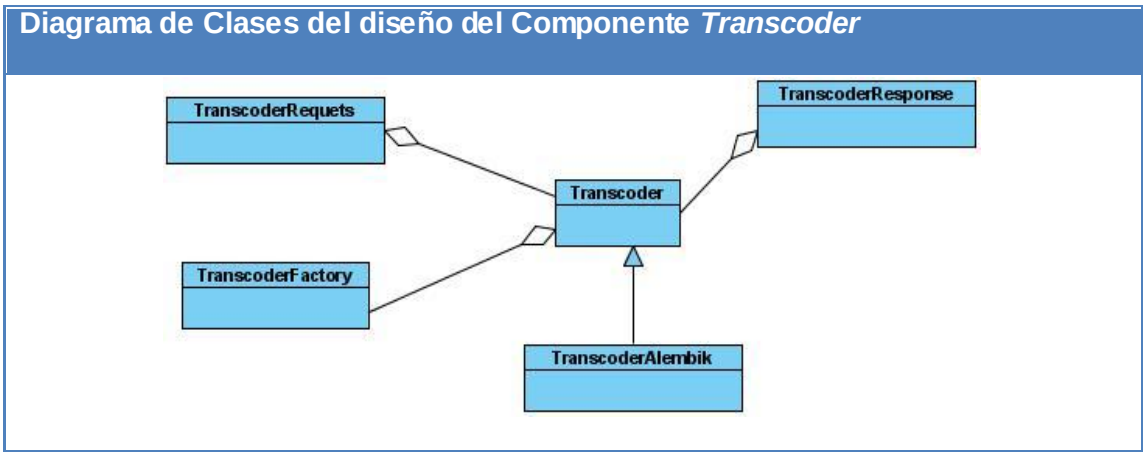


Figura 15: Diagrama de Clases del Diseño del Componente *Transcoder (46)*

Componente *Payment*

Este componente se dividió en seis paquetes para su mejor comprensión. A continuación se muestra su estructuración y el contenido de cada uno.

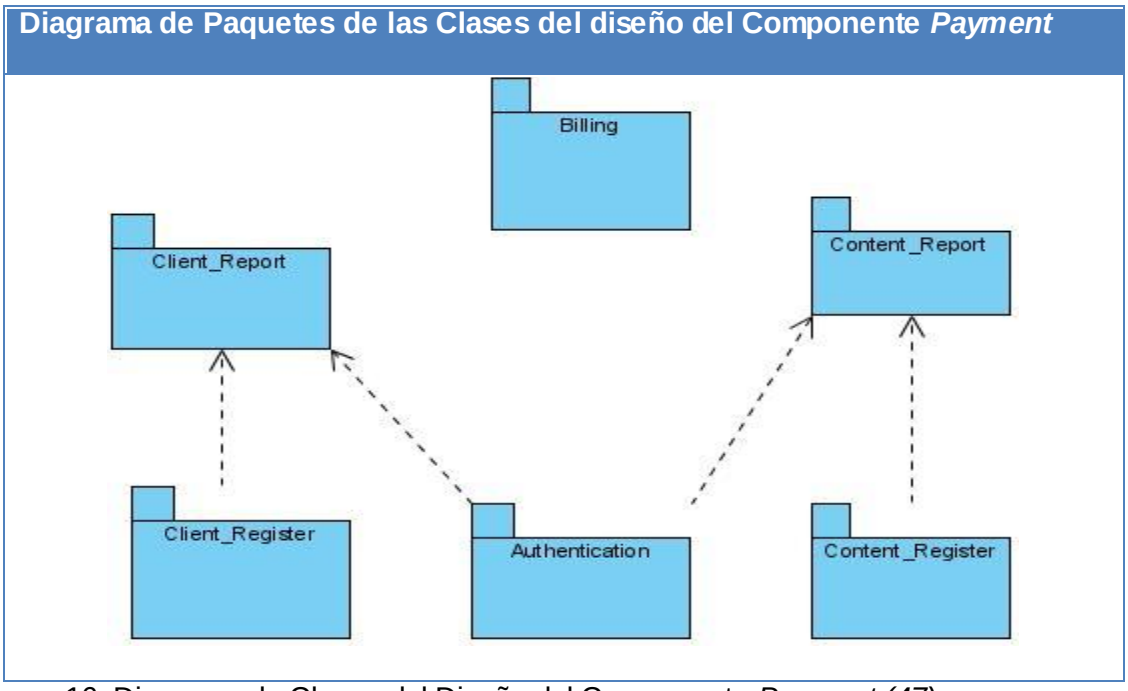


Figura 16: Diagrama de Clases del Diseño del Componente *Payment (47)*

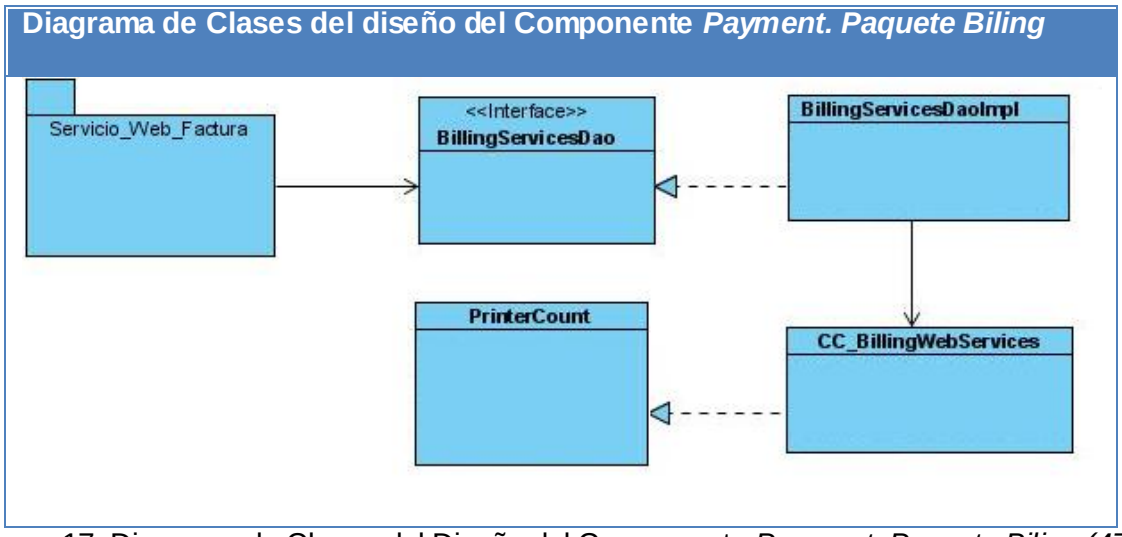


Figura 17: Diagrama de Clases del Diseño del Componente *Payment. Paquete Biling (47)*

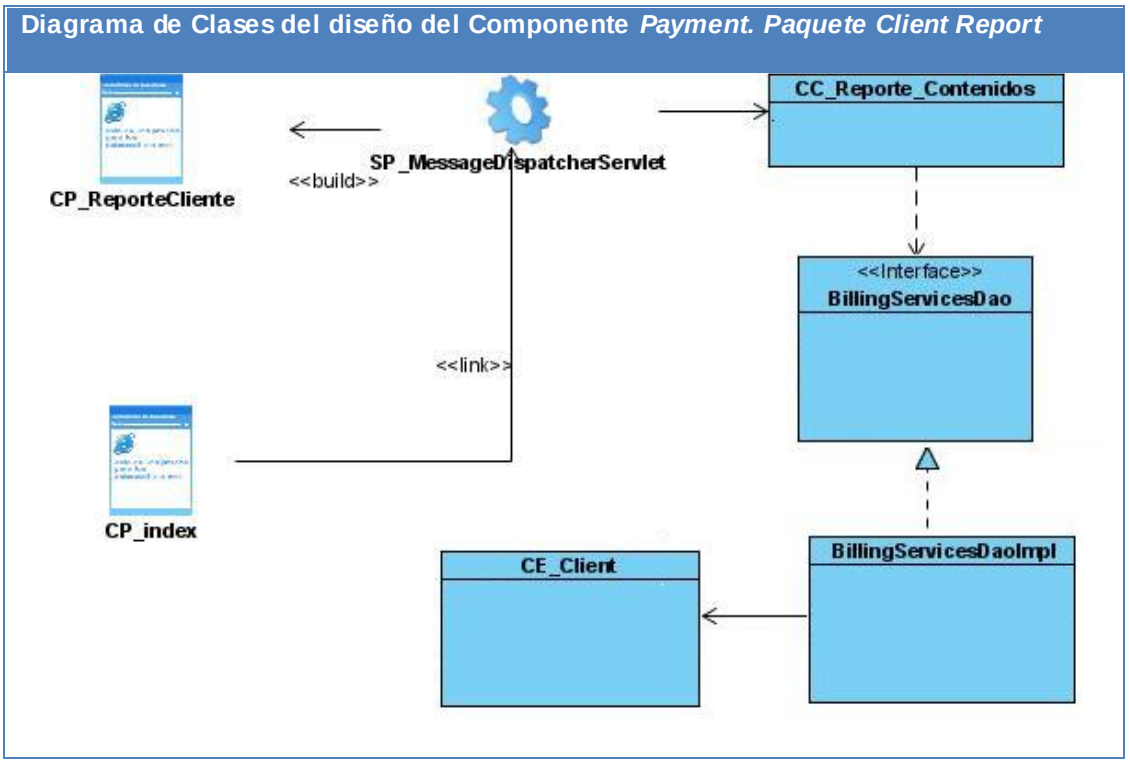


Figura 18: Diagrama de Clases del Diseño del Componente *Payment. Paquete Client Report (47)*

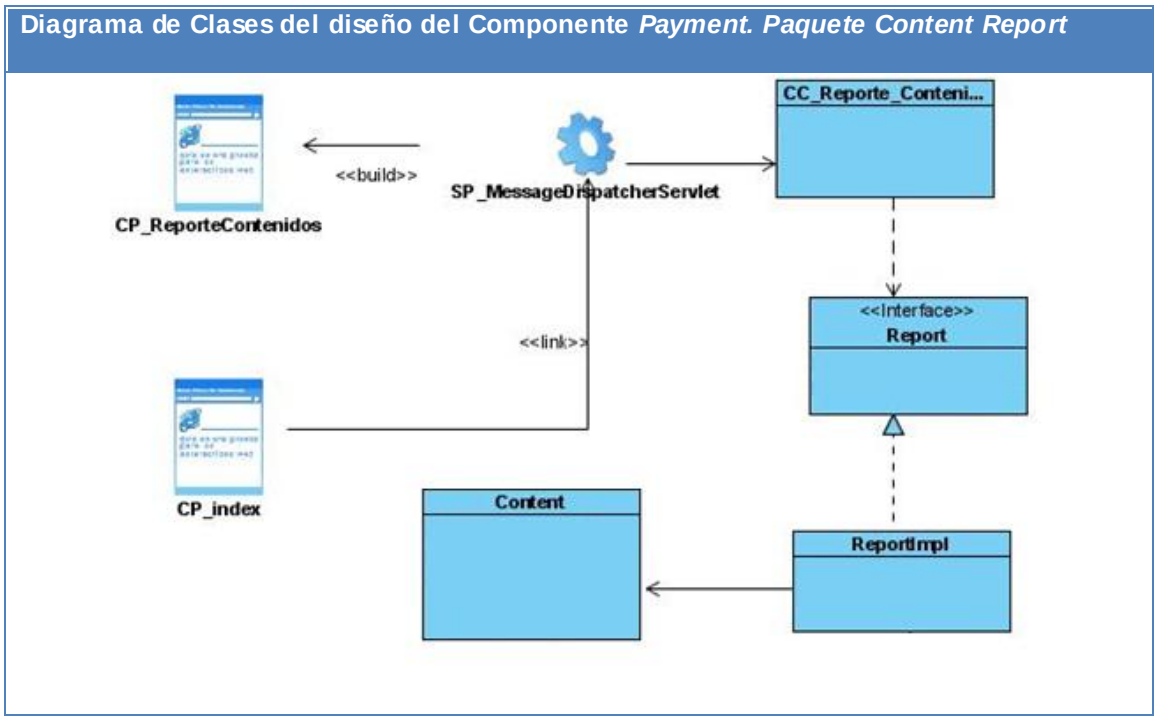


Figura 19: Diagrama de Clases del Diseño del Componente *Payment. Paquete Content Report (47)*

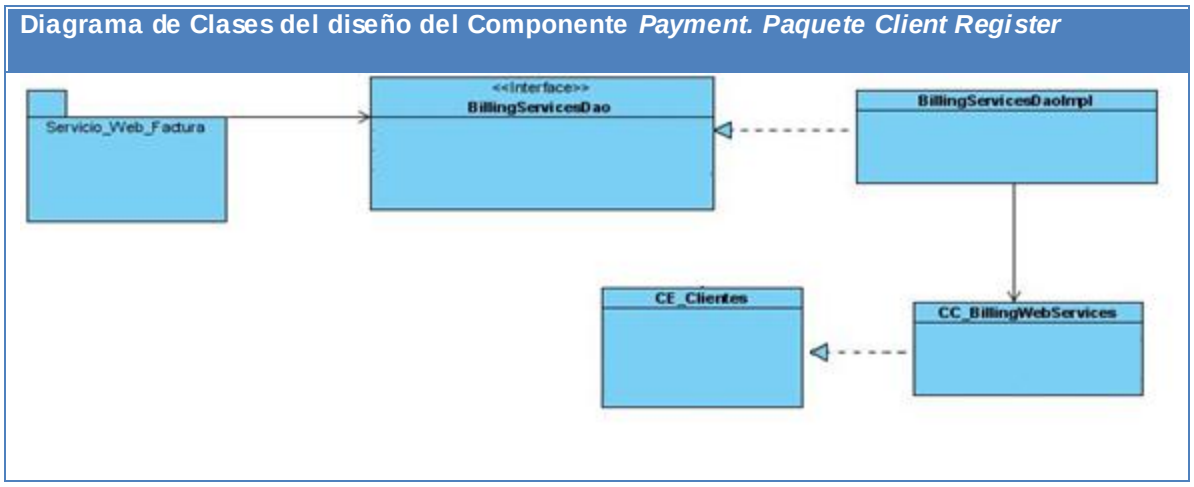


Figura 20: Diagrama de Clases del Diseño del Componente *Payment. Paquete Client Register (47)*

Diagrama de Clases del diseño del Componente *Payment. Paquete Content Register*

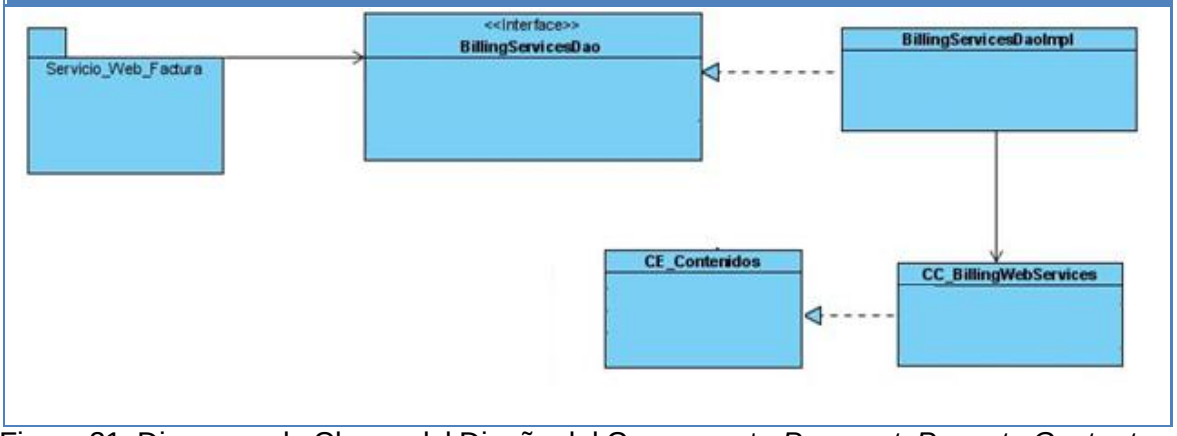


Figura 21: Diagrama de Clases del Diseño del Componente *Payment. Paquete Content Register (47)*

Diagrama de Clases del diseño del Componente *Payment. Paquete Authentication*

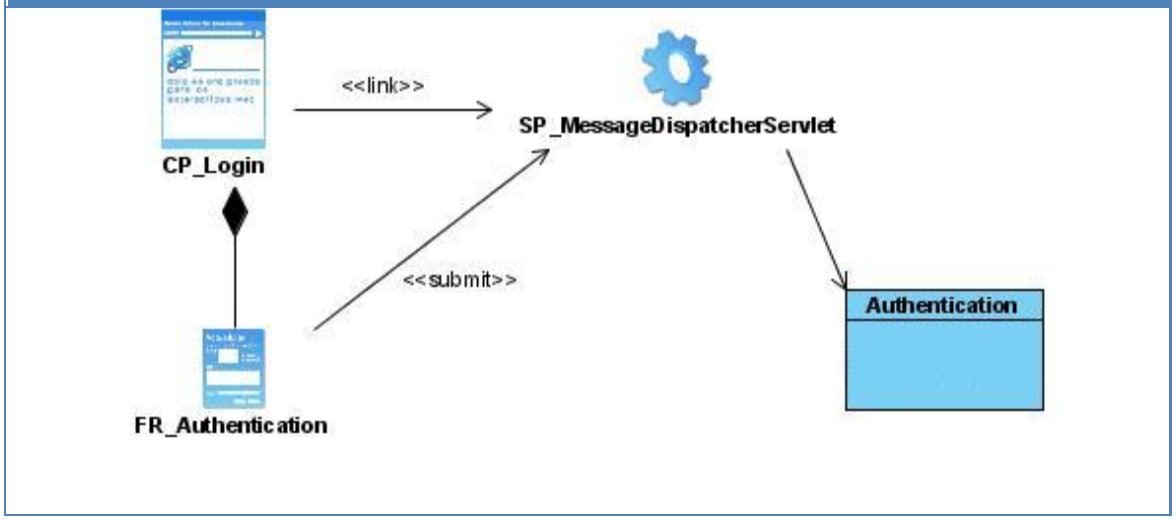


Figura 22: Diagrama de Clases del Diseño del Componente *Payment. Paquete Authentication (47)*

3.7 Vista de Implementación

En esta sección se describe la estructura general del modelo de implementación donde se representa la descomposición del *software*. A continuación se muestra el diagrama de componentes para cada uno de los módulos.

Componente *Content Delivery*

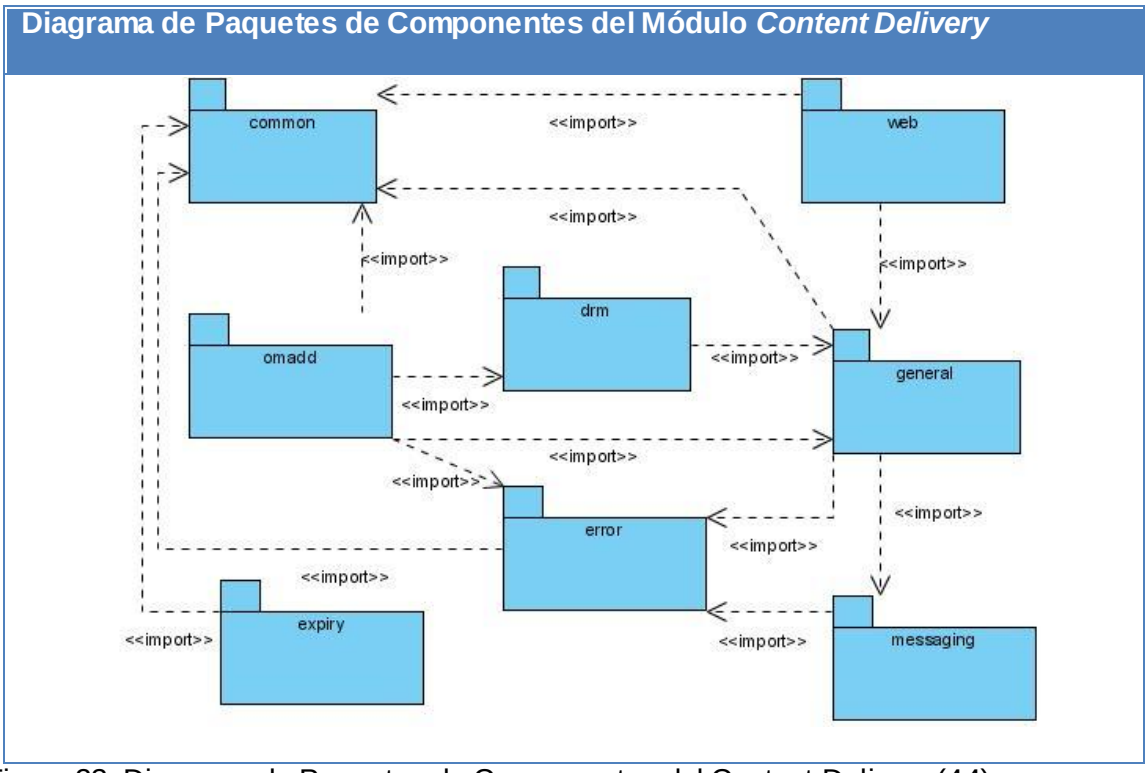


Figura 23: Diagrama de Paquetes de Componentes del Content Delivery (44)

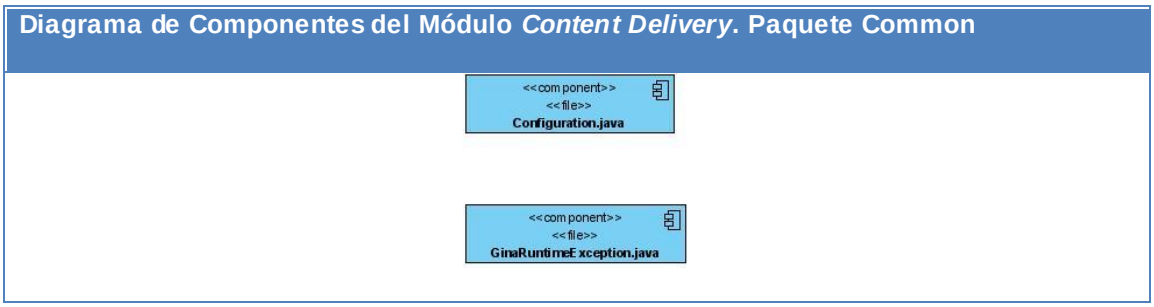


Figura 24: Diagrama de Componentes del Content Delivery. Paquete Common (44)

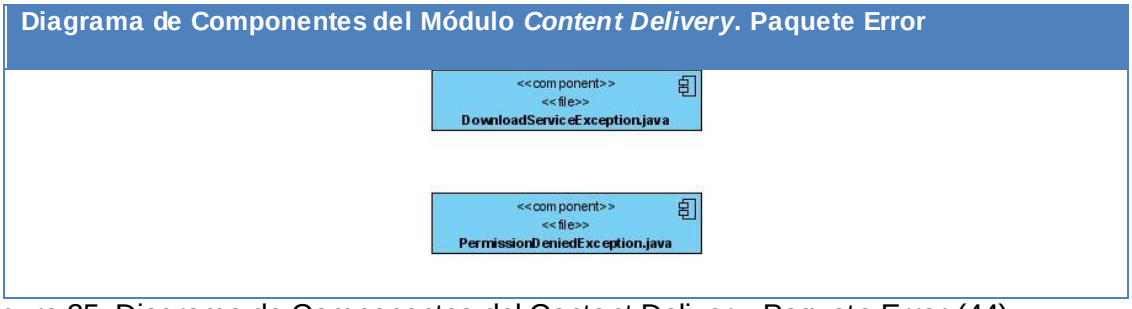


Figura 25: Diagrama de Componentes del Content Delivery. Paquete Error (44)

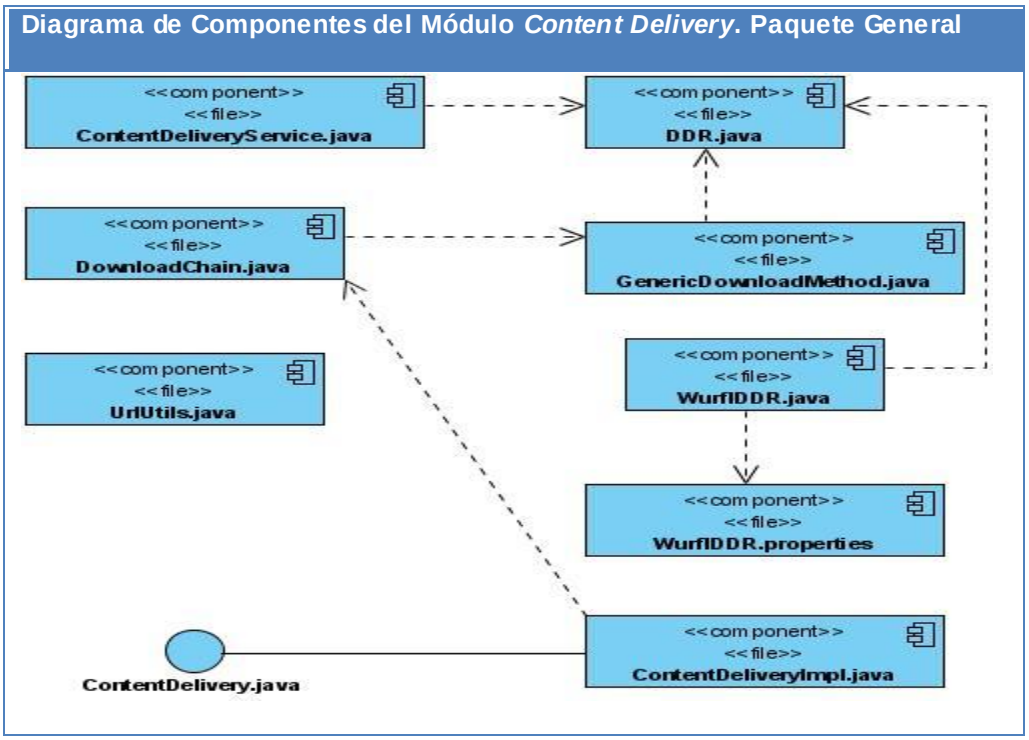


Figura 26: Diagrama de Componentes del Content Delivery. Paquete General (44)

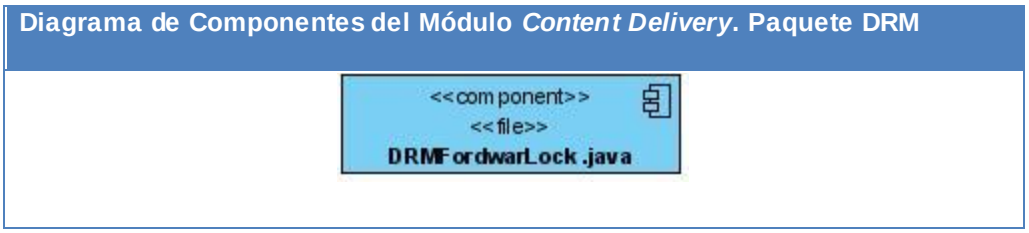


Figura 27: Diagrama de Componentes del Content Delivery. Paquete DRM (44)

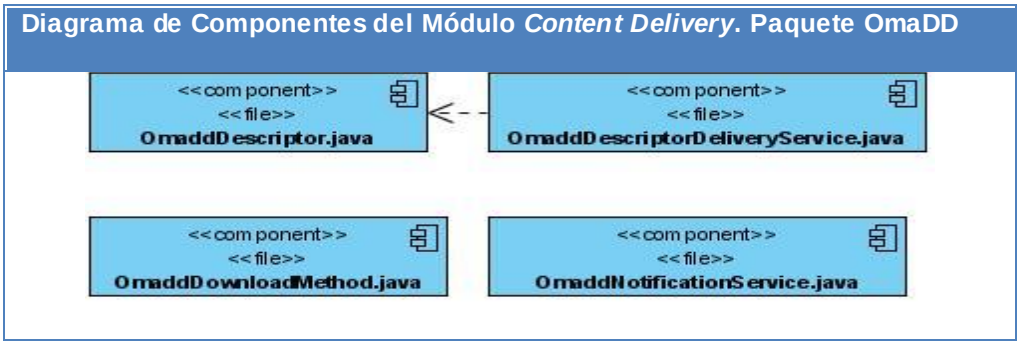


Figura 28: Diagrama de Componentes del Content Delivery. Paquete OmaDD (44)

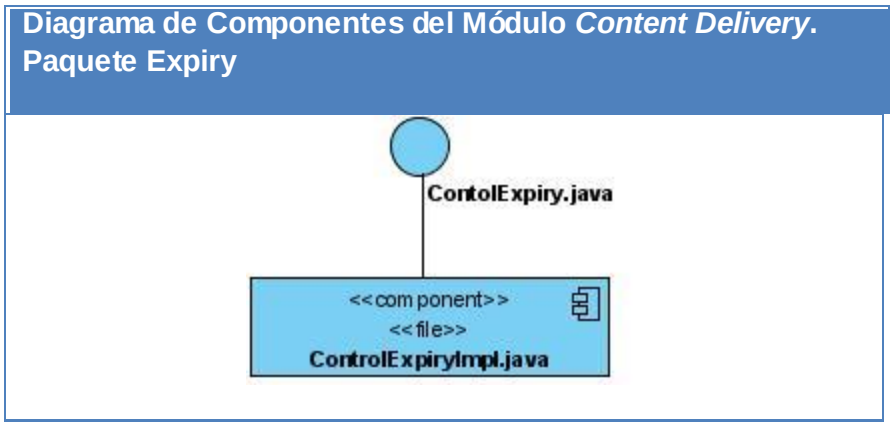


Figura 29: Diagrama de Componentes del Content Delivery. Paquete Expiry (44)

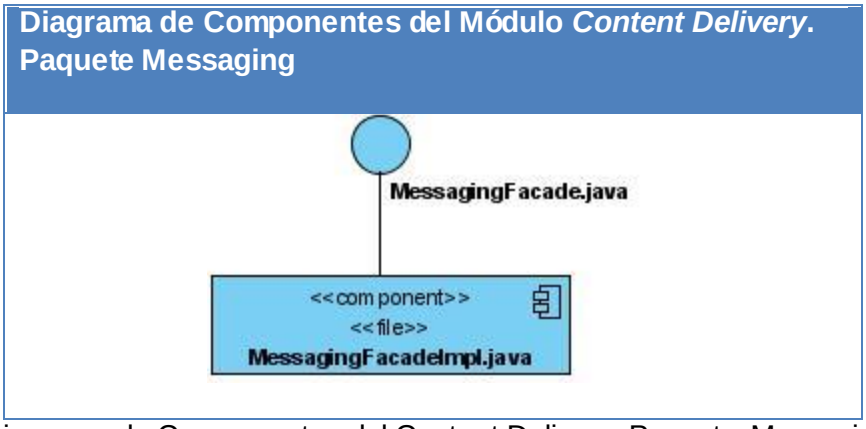


Figura 30: Diagrama de Componentes del Content Delivery. Paquete Messaging (44)

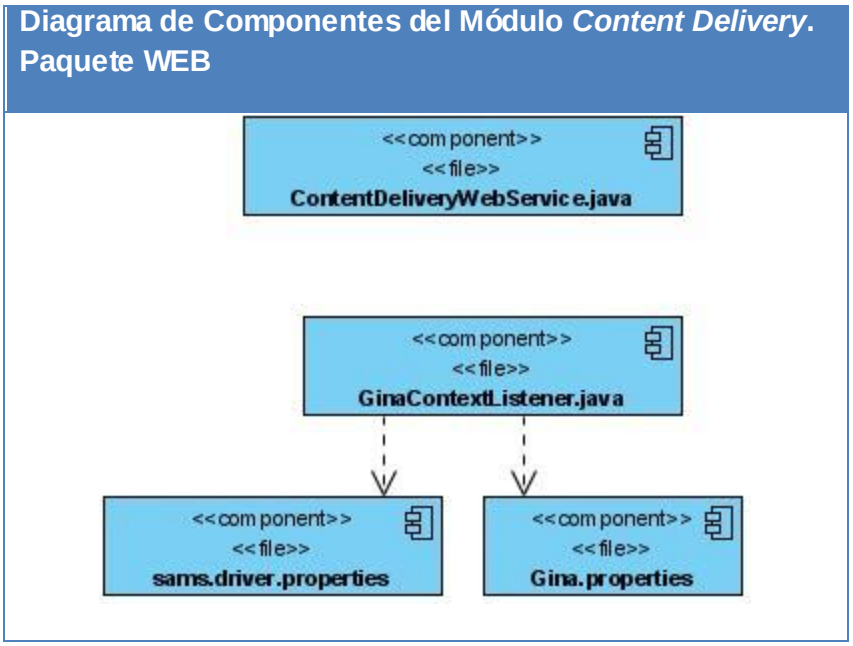


Figura 31: Diagrama de Componentes del Content Delivery. Paquete WEB (44)

Componente *Content*

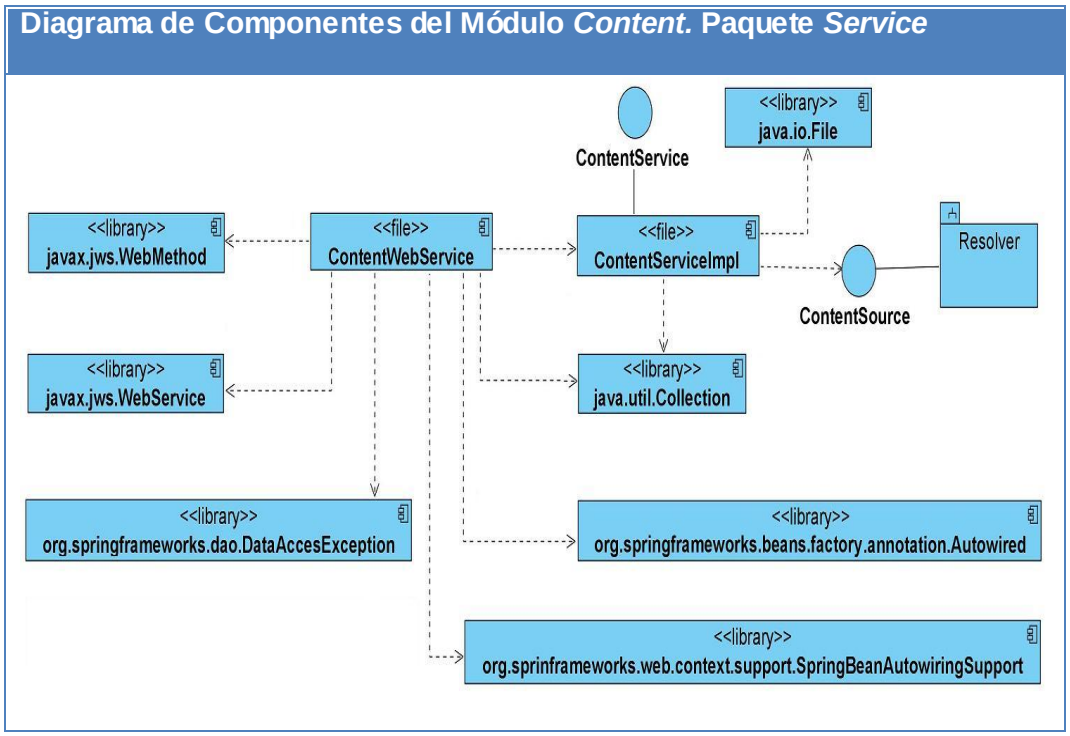


Figura 32: Diagrama de Componentes del Content. Paquete Service *(45)*

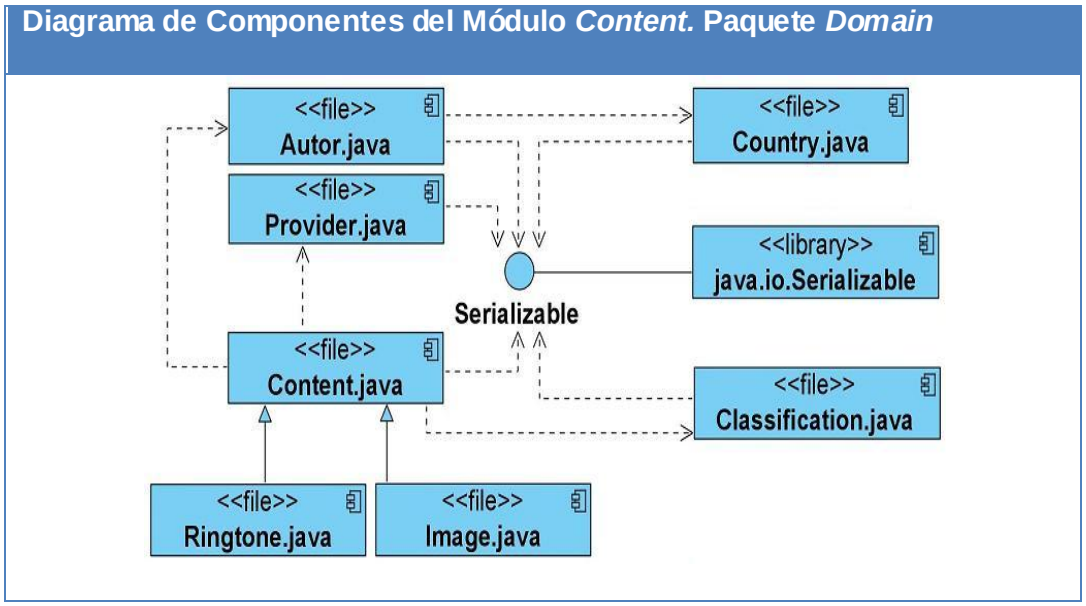


Figura 33: Diagrama de Componentes del Content. Paquete Domain *(45)*

Diagrama de Componentes del Módulo *Content.* Paquete *Resolver*

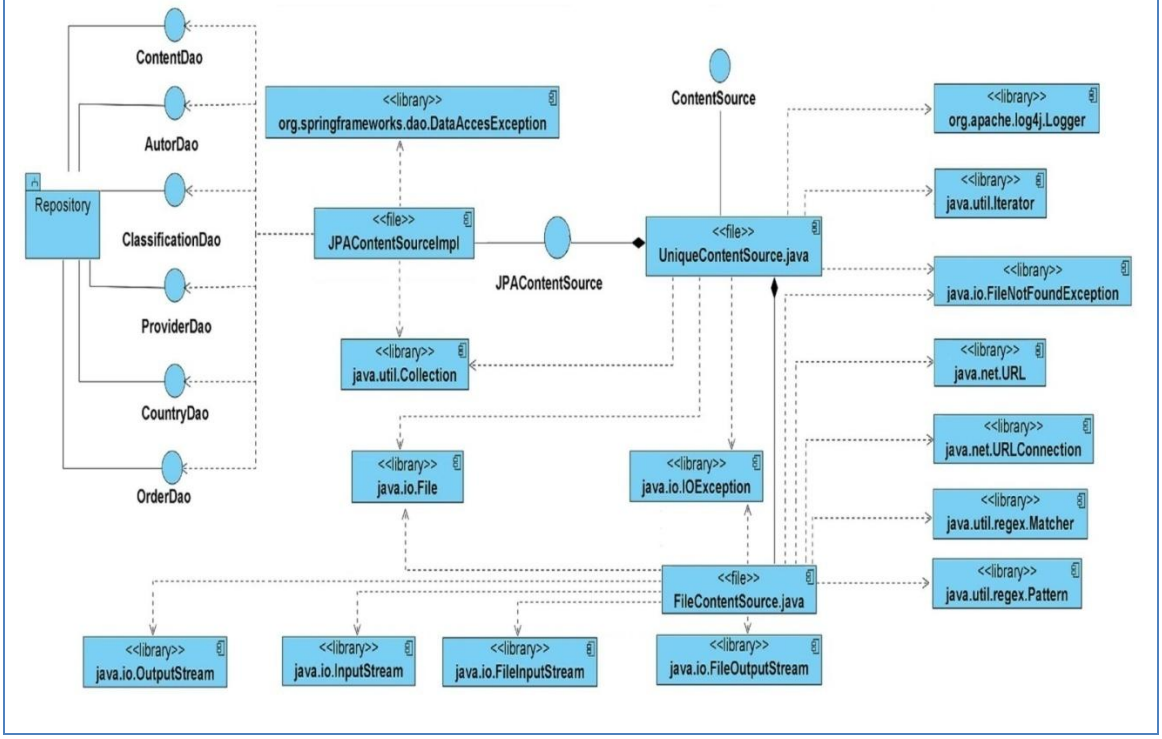


Figura 34: Diagrama de Componentes del Content. Paquete Resolver *(45)*

Diagrama de Componentes del Módulo *Content.* Paquete *Repository*

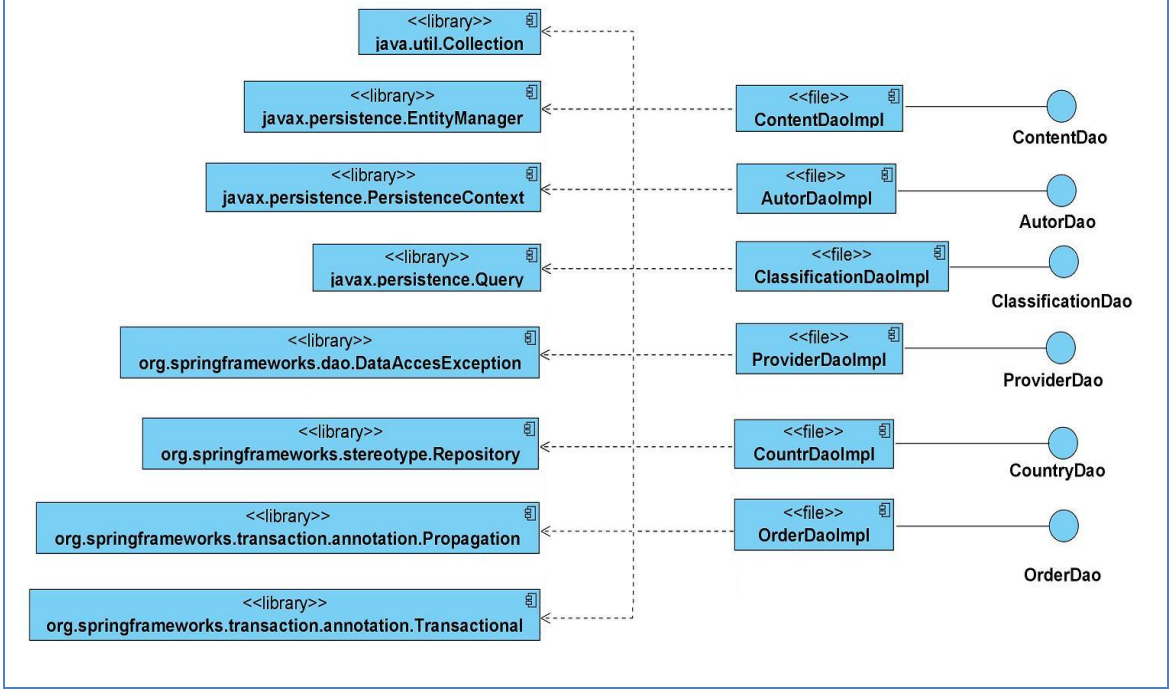


Figura 35: Diagrama de Componentes del Content. Paquete Repository *(45)*

Componente *Transcoder*

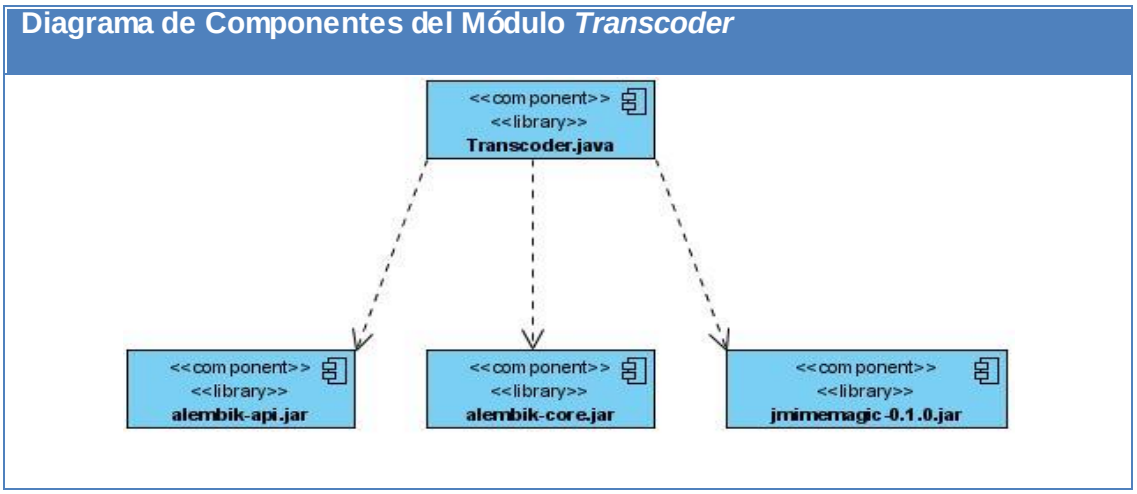


Figura 36: Diagrama de Componentes del Trancoder (46)

Componente *Payment*

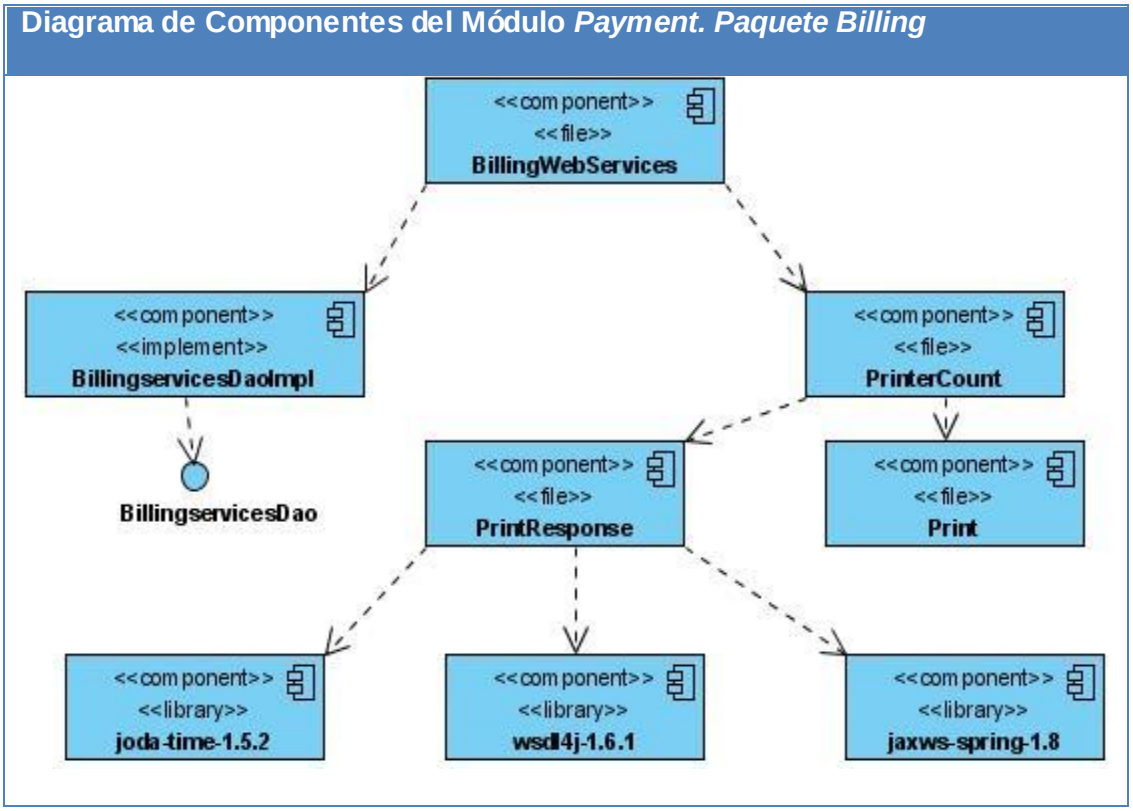


Figura 37: Diagrama de Componentes del Payment. Paquete Billing (47)

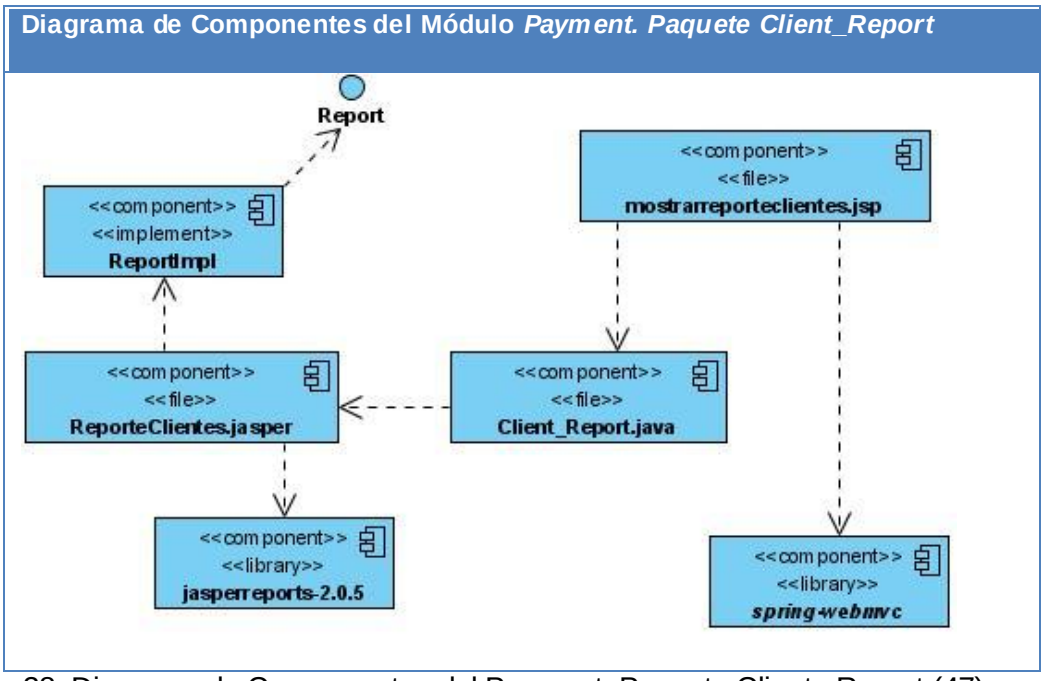


Figura 38: Diagrama de Componentes del Payment. Paquete Client_ Report (47)

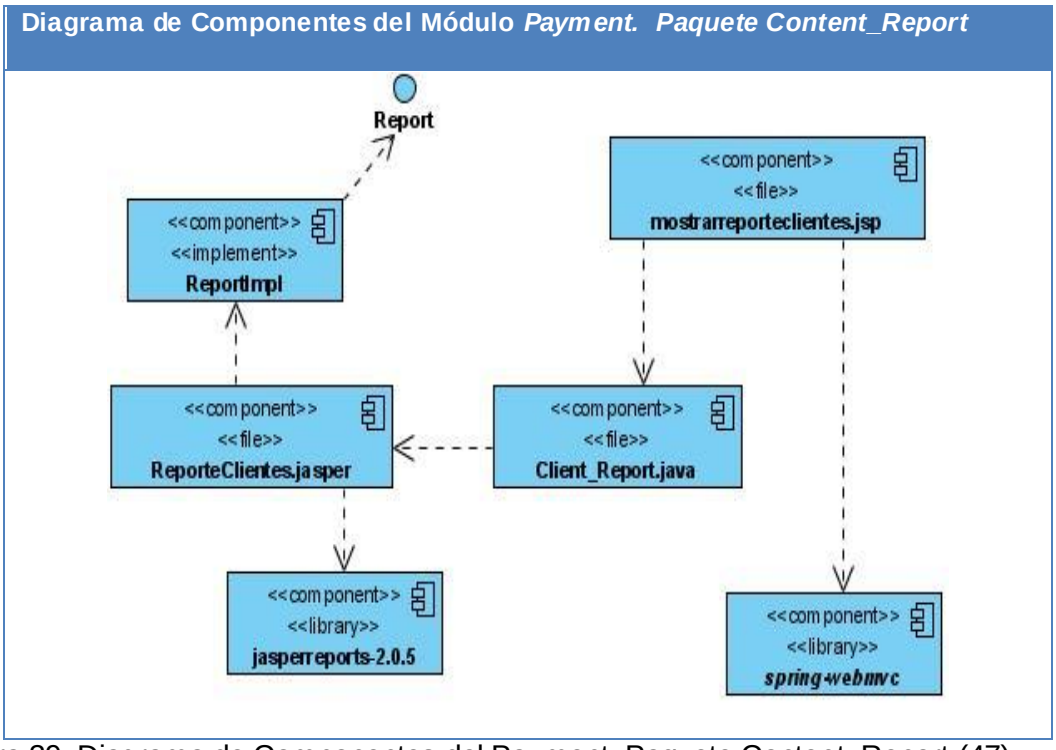


Figura 39: Diagrama de Componentes del Payment. Paquete Content_Report (47)

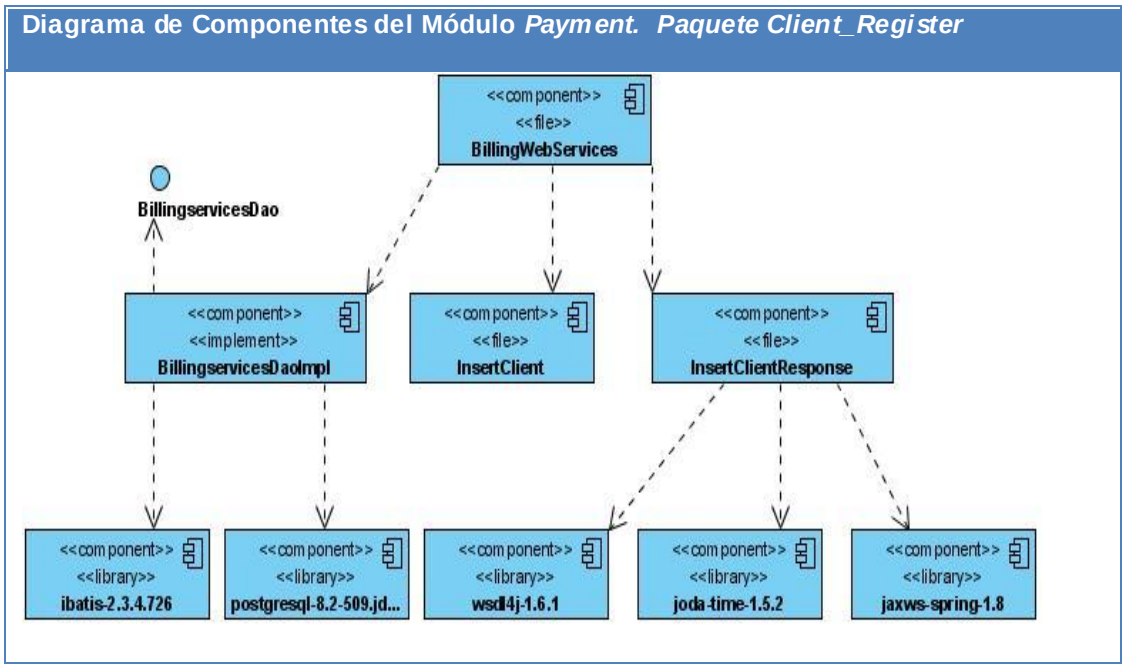


Figura 40: Diagrama de Componentes del Payment. Paquete Client_Register (47)

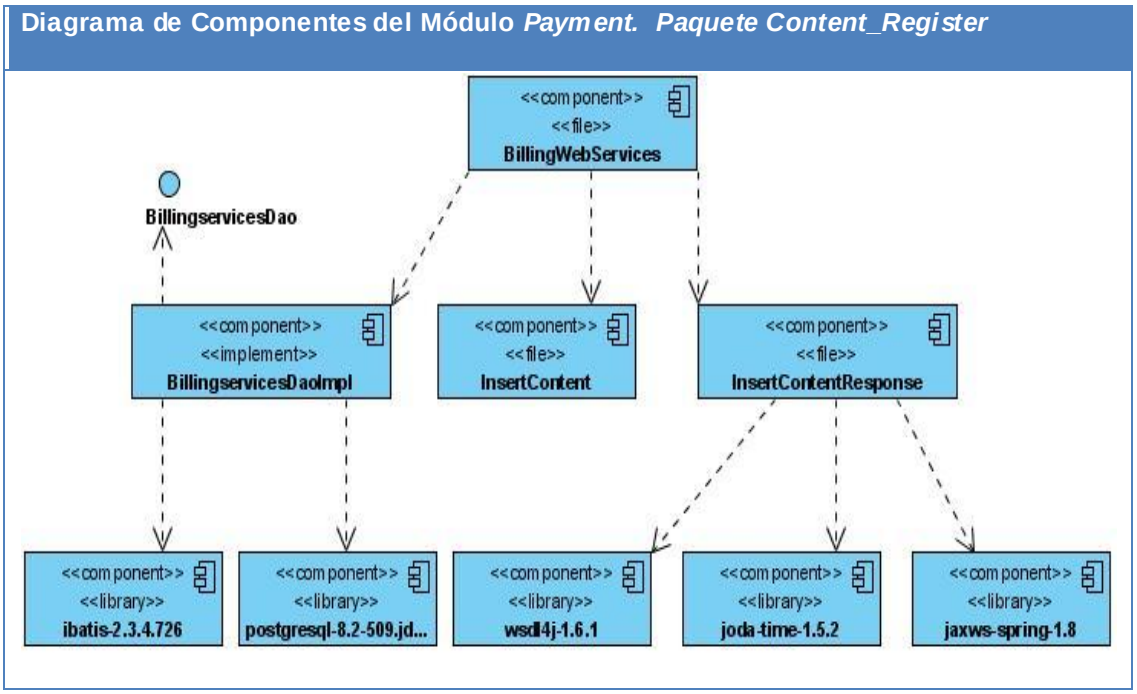


Figura 41: Diagrama de Componentes del Payment. Paquete Content_Register (47)

Diagrama de Componentes del Módulo *Payment. Paquete Authentication*

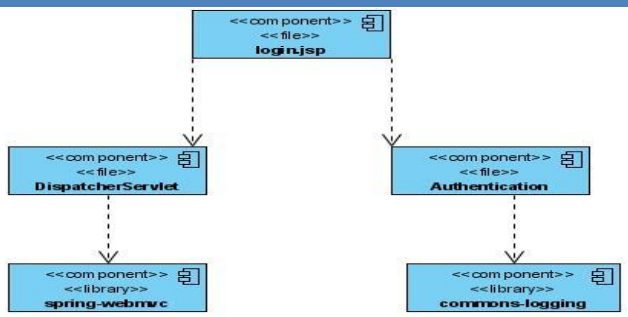


Figura 42: Diagrama de Componentes del Payment. Paquete Authentication (47)

3.8 Vista de Despliegue

La vista de despliegue suministra una base para la comprensión de la distribución física de un sistema a través de nodos; es decir, modela el funcionamiento del *software* en el *hardware* y especifica los protocolos de comunicación entre el *hardware*.

Diagrama de Despliegue

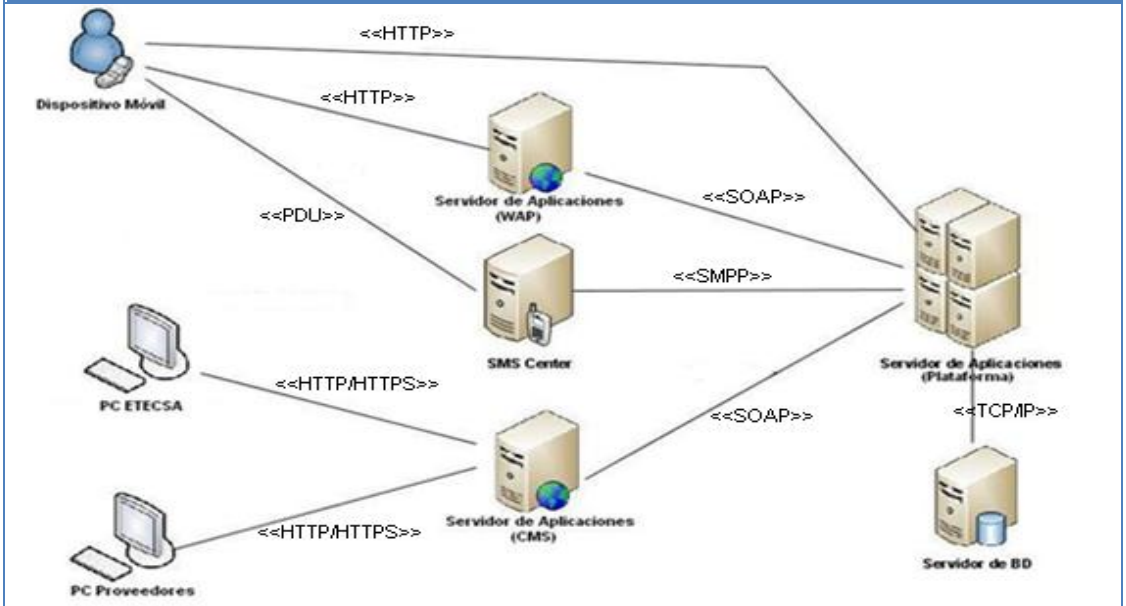


Figura 43: Diagrama de Despliegue de la Plataforma.

Descripción de los nodos:

Dispositivo Móvil: Representa a los usuarios finales de la plataforma, los cuales se conectan al portal WAP y a la plataforma mediante HTTP y al SMS Center con PDU.

PC ETECSA: Representan al personal autorizado de ETECSA[8] que realizan tareas de administración, se conectan a la interfaz de administración en el servidor de aplicaciones (CMS) a través de HTTP, aunque el envío de contraseñas se hace a través de HTTPS.

PC Proveedores: Representan a los proveedores que gestionan los contenidos para la plataforma y se conectan a la interfaz de administración en el servidor de aplicaciones (CMS) a través de HTTP, aunque el envío de contraseñas se hace a través de HTTPS.

Servidor de Aplicaciones (WAP): En este servidor se encuentra el Portal WAP, por el cual navegan los usuarios finales para obtener noticias y llegar a la plataforma de gestión de contenido. Interactúa con los ordenadores a través de HTTP y con el servidor de aplicaciones donde está la plataforma con SOAP.

Servidor de Aplicaciones (CMS): En este servidor se encuentra la interfaz de administración por la cual los proveedores y el personal autorizado de ETECSA pueden interactuar con los diferentes parámetros de configuración y con los contenidos de la plataforma y del portal WAP. Interactúa con los ordenadores a través de HTTP y con el servidor de aplicaciones donde está la plataforma con SOAP.

SMS Center: Representa un elemento de la red de telefonía móvil cuya función es la de enviar/recibir mensajes SMS. Se comunica con los dispositivos móviles a través del protocolo PDU y con el servidor de aplicaciones donde está la plataforma con SMPP.

Servidor de Aplicaciones (Plataforma): Constituye el nodo fundamental, aquí se encuentra la plataforma de Gestión de Contenidos para Dispositivos Móviles y de alguna manera todos los nodos se conectan con él, aunque no sea directamente.

Servidor de Base de Datos: Este nodo guarda la base de datos, con todos los contenidos y reportes de la plataforma.

Descripción de los protocolos de comunicación:

HTTP: La transferencia de hipertexto (*HyperText Transfer Protocol*) es el protocolo usado en cada transacción de la *web*. Se utiliza para comunicar los dispositivos móviles con los servidores de aplicaciones (WAP) y el de la plataforma, además une a la PC_ETECSA y a la PC_Proveedores con el servidor de aplicaciones (CMS).

[8]ETECSA: Es la Empresa de Telecomunicaciones de Cuba, una organización de capital mixto que tiene como objeto social prestar los servicios públicos de telecomunicaciones.

HTTPS: La transferencia segura de hipertexto (*Hypertext Transfer Protocol Secure*), es un protocolo de red basado en el protocolo HTTP, destinado a la transferencia segura de datos de hipertexto, es decir, es la versión segura de HTTP. Se usa para conectar el servidor de aplicaciones de la plataforma con los servidores de aplicaciones (WAP) y (CMS) para asegurar la transferencia de contraseñas.

TCP/IP: En referencia a los dos protocolos más importantes que la componen: Protocolo de Control de Transmisión (TCP) y Protocolo de Internet (IP), que fueron los dos primeros en definirse, y que son los más utilizados mundialmente. La familia de protocolos TCP/IP es la base de Internet, y sirve para enlazar computadoras.

SMPP: El envío de mensajes cortos punto a punto (S*hort Message Peer-to-peer Protocol*), es un protocolo estándar de telecomunicaciones pensado para el intercambio de mensajes SMS entre equipos que gestionan los mensajes y se utiliza normalmente para permitir a terceros enviar mensajes. Conecta los nodos del servidor de aplicaciones donde está la plataforma con el SMS Center.

PDU: El protocolo de unidad de datos (*Protocol Data Unit*), transfiere del nodo origen al nodo destino, el mensaje sin cabecera al usuario. Se usa para conectar el SMS Center con los dispositivos móviles de los usuarios finales.

SOAP: El protocolo de acceso a datos (*Simple Object Access Protocol*), define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. SOAP es uno de los protocolos más utilizados en los servicios Web.

3.9 Vista de datos

La necesidad de definir una estructura que permita almacenar datos, reconocer el contenido, y recuperar la información hacen de la vista de datos una perspectiva esencial en el desarrollo del *software*. Dicha vista incluye el modelado de los datos sobre las entidades que procesará el sistema, los atributos y las relaciones entre las mismas, a través del diagrama de clases persistentes y del de Entidad-Relación 45).

Modelo Lógico de Datos.

Diagrama de Clases Persistentes

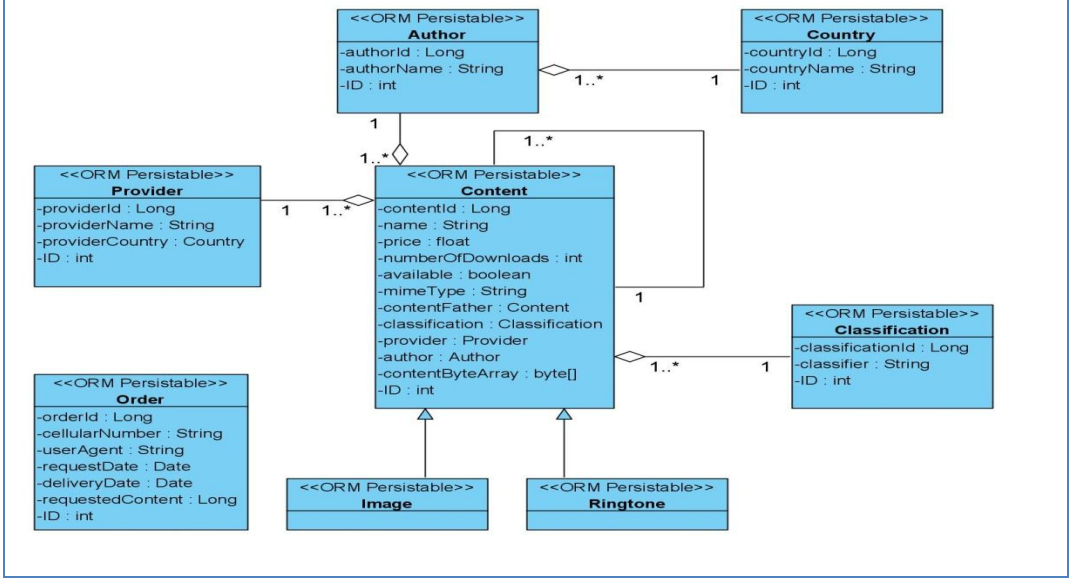


Figura 44: Diagrama de Clases Persistentes.

Modelo Físico de Datos.

Diagrama de Entidad-Relación

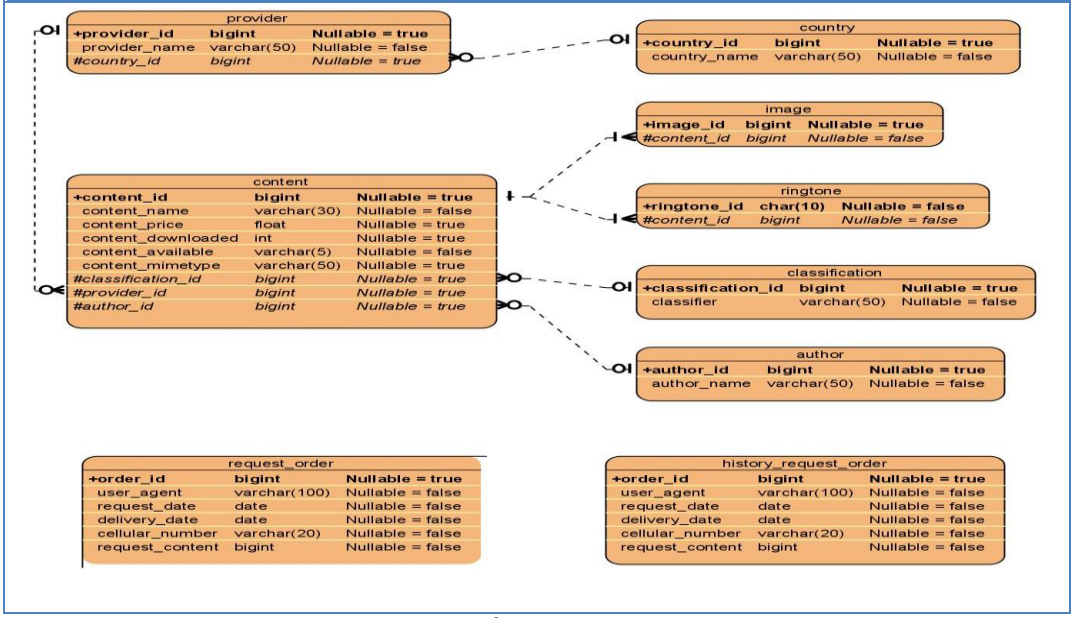


Figura 45: Diagrama Entidad-Relación.

3.10 Conclusiones

En este apartado se definieron las características más importantes de la arquitectura, se describieron los componentes que la integran, el flujo de procesos a través de dichos componentes y las vistas arquitectónicas que expresan detalladamente su estructura.

Capítulo 4: Evaluación de la arquitectura

4.1 Introducción

Todo diseño arquitectónico debe tener en cuenta los atributos de calidad. La mayoría de estos se miden por los requisitos no funcionales, los cuales definen los escenarios, tácticas a utilizar y *frameworks* de desarrollo, todo en función de satisfacer dichos atributos.

En esta sección se realiza una descripción de cómo la arquitectura del *software* va a contribuir al desarrollo de todas las capacidades del Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles.

4.2 Evaluando una Arquitectura

4.2.1 Necesidad de evaluar una arquitectura de *Software*

Con la evaluación de la arquitectura de *software* podemos prevenir todas las posibles fisuras de un diseño que no cumpla con los requerimientos de calidad y saber que tan adecuada es para el sistema.

La evaluación de una arquitectura no dice si es buena o mala, o da una calificación; pero sí expresa donde está el riesgo, al identificar las fortalezas y debilidades con que cuenta.

Después de la evaluación de una arquitectura de *software*, se pueden tomar algunas decisiones como: si se puede seguir el proyecto con las áreas de debilidad dadas en la evaluación, si hay que reforzar la arquitectura o si hay que comenzar de nuevo.

4.2.2 Momentos de evaluación de una arquitectura de Software

La evaluación clásica de la arquitectura se realiza cuando esta se encuentra explicada totalmente y no se ha iniciado su implementación. No obstante, se puede evaluar en cualquier momento del desarrollo.

Existen dos variantes útiles para realizar una evaluación: la temprana y la tardía (48).

La evaluación temprana

Para realizar esta evaluación no es necesario que la arquitectura se encuentre completamente especificada. Ella permite efectuar decisiones sobre la arquitectura en cualquier nivel, puesto que se pueden imponer cambios arquitectónicos producto de una evaluación, en función de los atributos de calidad esperados.

La evaluación tardía

Se realiza cuando la arquitectura del sistema se encuentra establecida y se ha terminado su implementación; es decir, en el momento de adquisición de un sistema ya terminado. Consideran los autores que la evaluación en este punto es muy importante y útil, puesto que puede observarse el cumplimiento de los atributos de calidad asociados al sistema y cómo será su comportamiento general.

La evaluación de la arquitectura de *software* debe realizarse cuando esta contiene suficientes elementos como para justificarla. Un buen momento para determinar cuándo realizar la evaluación es cuando el equipo de desarrollo comienza a tomar decisiones que dependen de la arquitectura, y que de no tenerlas en cuenta, aumentaría el costo de realizar una evaluación.

4.2.3 Personas Involucradas en la evaluación

Habitualmente, las evaluaciones son realizadas por miembros del equipo de desarrollo, por ejemplo el arquitecto, el diseñador y el líder de proyecto. Aunque pueden existir excepciones, en las que las pruebas son realizadas por un grupo de especialistas.

El cliente también se interesa por las evaluaciones, pues en dependencia de los resultados obtenidos durante las pruebas, puede decidir si se continúa o no con el proyecto.

4.2.4 Resultado de la evaluación

El resultado de la evaluación de una arquitectura de *software* no es cuantitativo, sino que produce un informe, que varía de acuerdo al método utilizado, para plasmar las debilidades encontradas.

Este informe se centra en decir si esa arquitectura es adecuada y en caso de que existan varias propuestas, comunicar cuál es la óptima para el sistema.

No es de interés conocer la cantidad de transacciones por segundos en la evaluación arquitectónica. Lo importante es verificar cómo un atributo de calidad es afectado por una decisión de diseño arquitectónico.

Una arquitectura es adecuada si el sistema resultante cumple con los objetivos de calidad y si este puede ser construido con los recursos disponibles (49).

4.3 Atributos de la calidad

La calidad del *software* es una preocupación a la que se dedican muchos esfuerzos. Sin embargo, el sistema casi nunca es perfecto. Todo proyecto tiene como objetivo producir *software* de la mejor calidad posible que cumpla y, si puede, supere las expectativas de los usuarios.

Para ello, el sistema debe cumplir con algunos requerimientos adicionales, que en gran medida ayudan a mejorar el trabajo realizado. Estos requisitos se conocen con el nombre de atributos de calidad.

A continuación se listan algunas de las cualidades por la que se puede evaluar una arquitectura de *software*.

Seguridad. Es la habilidad del sistema para resistir a intentos de uso no autorizados y denegación del servicio, mientras se sirve a usuarios legítimos.

Eficiencia. Es la habilidad del sistema de hacer el trabajo para el cual fue construido en el tiempo previsto.

Disponibilidad. Se refiere al tiempo en que el sistema está disponible a las necesidades de los usuarios.

Mantenibilidad. Es la factibilidad de dar mantenimiento al sistema.

Confiabilidad. Es la medida del sistema para intercambiar datos de una forma segura con los usuarios que hacen las peticiones.

Portabilidad. Es la habilidad del sistema de ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser *hardware*, *software* o una combinación de ambos.

Modificabilidad. Es la posibilidad de realizar cambios al sistema en forma rápida y a bajo costo.

Fiabilidad. Está relacionada con la capacidad para reaccionar frente a fallos internos o externos o ante cualquier anomalía.

Reusabilidad. Es la capacidad de la arquitectura de ser expandida o modificada para producir otras nuevas. Es importante cuando la arquitectura se va a utilizar como elemento fundamental de toda una familia de productos relacionados.

Interoperatibilidad. Es la condición mediante la cual el sistema puede intercambiar procesos o datos con otras aplicaciones.

4.4 Técnicas para evaluar la arquitectura de *software*

Existen varias técnicas que permiten realizar evaluaciones a la arquitectura, estas se clasifican en cualitativas y cuantitativas.

Por lo regular, las cualitativas son usadas cuando la arquitectura se encuentra en construcción, mientras que las cuantitativas, se usan cuando la arquitectura ya ha sido implantada.

En las evaluaciones cualitativas se pueden utilizar escenarios, cuestionarios o listas de verificación; mientras que en las cuantitativas se pueden emplear métricas, simulaciones, prototipos, experimentos o modelos matemáticos. A continuación se explican algunas de ellas (49).

4.4.1 Evaluación basada en escenarios

Un escenario es una secuencia específica de pasos que involucran el uso del sistema. Puede verse como la interacción de un actor con el sistema y este cuenta de tres partes (estímulo, contexto y respuesta). El estímulo muestra la interacción con el sistema (ejecución de tareas, cambio de configuración). El contexto describe qué sucede en el sistema ante un determinado estímulo. La respuesta explica, a través de la arquitectura, cómo debe responder el sistema ante el estímulo, lo que posibilita asociarlo con determinado atributo de calidad. La evaluación basada en escenarios entra en el grupo de las técnicas de evaluación cualitativas.

4.4.2 Evaluación basada en simulación

Consiste en la evaluación del comportamiento de la propuesta arquitectónica bajo determinadas circunstancias. Para ello se utilizan componentes implementados con cierto nivel de abstracción, que simulan la ejecución de lo que sería la arquitectura en realidad. La evaluación basada en simulación es una de las técnicas de evaluación cuantitativas.

4.4.3 Evaluación basada en experiencia

Como su nombre lo que indica, esta técnica se orienta fundamentalmente al conocimiento que pueden aportar los especialistas durante el desarrollo del proyecto. Estos conocimientos constituyen, por lo general, valiosas herramientas en la toma de decisiones acertadas. La evaluación basada en experiencia pertenece a las técnicas de evaluación cualitativas.

4.5 Métodos para evaluar la calidad de la arquitectura

Los Métodos para evaluar las arquitecturas de *software* no están implantados desde hace mucho tiempo. En virtud de esto se han propuesto múltiples métodos de evaluación tales como: *Software Architecture Analysis Method* (SAAM), *Architecture Trade-off Analysis Method* (ATAM), *Active Reviews for Intermediate Designs* (ARID), *Cost-Benefit Analysis Method* (CBAM), entre otros.

La arquitectura de *software* posee un gran impacto sobre la calidad de un sistema, de ahí que es de vital importancia la evaluación de la misma (49).

A continuación se explican los métodos más utilizados a nivel mundial.

4.5.1 SAAM (Método de Análisis de Arquitectura de software)

El método SAAM está basado en escenarios, lo que permite la evaluación de atributos de calidad y la relación del diseño arquitectónico con los requerimientos del sistema. SAAM posibilita evaluar múltiples arquitecturas aunque en ningún caso emite un valor absoluto referente a la calidad del sistema.

4.5.2 ARID (Método de Revisiones Activas para Diseños intermedios)

El método de Revisiones Activas para Diseños Intermedios permite realizar evaluaciones a diseños parciales en etapas tempranas dentro del desarrollo del proyecto (50). No se focaliza en los atributos de calidad, sino en el análisis de los componentes independientes y se aplica durante todo el diseño de la arquitectura.

4.5.3 ATAM (Método de Análisis de Arquitecturas de Intercambio)

El método ATAM representa la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo interactúan y los tipos de acuerdos que se establecen entre ellos.

El método se concentra en la identificación de los estilos arquitectónicos o enfoques utilizados. De cualquier forma, estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros.

4.5.4 Comparación entre métodos de evaluación

A continuación se presenta una comparación entre los métodos de evaluación de arquitectura de software SAAM, ATAM y ARID.

Tabla 2: Comparativa entre los métodos SAAM, ATAM, ARID (49).

	ATAM	SAMM	ARID
Atributos de calidad Contemplados	-Modificabilidad -Seguridad -Confiabilidad -Desempeño	-Modificabilidad -Funcionabilidad	-Conveniencia del diseño evaluado
Objetos Analizados	-Estilos Arquitectónicos -Documentación -Flujo de Datos -Vistas Arquitectónicas	-Documentación -Vistas Arquitectónicas	-Especificación de los componentes.
Etapas del proyecto en las que se aplica	-Luego de que el diseño de la arquitectura ha sido establecido	-Luego de que al arquitectura cuenta con funcionabilidad ubicada en módulos	-A lo largo del diseño de la arquitectura

4.6 Evaluación de la arquitectura de *software* propuesta

La arquitectura propuesta para el Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles se valida evaluando cómo las vistas arquitectónicas responden a cada uno de los principales requerimientos y restricciones del sistema, a través del Método de Análisis de Acuerdos de Arquitectura (ATAM).

En conjunto con las arquitecturas de *software* y su evaluación, son importantes las propiedades o requerimientos no funcionales (RNF) del sistema de *software*, pues estas influyen notablemente en su calidad. Estos requerimientos no funcionales tienen un gran impacto en el desarrollo y mantenimiento del sistema, su operabilidad y el uso que este haga de los recursos.

Entre los RNF más importantes se encuentran: modificabilidad, eficiencia, mantenibilidad, interoperabilidad, confiabilidad, reusabilidad y seguridad (48).

Posteriormente se presenta de qué forma la arquitectura descrita cumple con dichos requerimientos.

Seguridad: Se maneja a través del canal seguro (HTTPS) para la transmisión de contraseñas, además de ello se agrega la seguridad a nivel de mensajes utilizando ws-security para toda la comunicación externa al Núcleo de la Plataforma; y todos los errores que se produzcan generarán trazas (Logs) de forma tal que puedan ser observados por los administradores del sistema.

Eficiencia: La mayor carga de procesamiento de la plataforma se encuentra en el componente Transcoding, debido a que en ocasiones el cambio de un formato a otro de un determinado tipo de contenido pudiera convertirse en una tarea compleja. Esto se garantiza con los niveles de concurrencia aceptados y que dependen, en gran medida, del *hardware* sobre el cual esté corriendo el sistema. No obstante, estos son configurables en cuanto al número de peticiones concurrentes que pueden ser procesadas por cada uno de los diferentes métodos de descarga que se implementen en la solución, si siguieran llegando peticiones, estas se almacenarían en una cola hasta que puedan ser atendidas.

Disponibilidad: En cuanto a parámetros de software se refiere, el sistema estará disponible a las necesidades de los usuarios las 24 horas del día, y los cambios y configuraciones se deberán hacer en tiempo de ejecución.

Mantenibilidad: El sistema presenta una arquitectura basada en componentes, de forma tal de que cada componente puede ser actualizado y mantenido de manera independiente.

De igual forma, para el desarrollo de la plataforma, se estableció un estándar de codificación que es utilizado por todos los desarrolladores. Esto tiene como objetivo garantizar un estilo de codificación uniforme en todo el sistema. Dicho documento entre

otras cosas, plantea que todo el código debe ser documentado para facilitar las labores de mantenimiento.

Además de ello, debido a la metodología de desarrollo seleccionada (RUP), se cuenta con suficiente documentación que posibilita dar mantenimiento con relativa facilidad.

Confiabilidad: La información que se intercambia con el usuario no contiene datos confidenciales. No obstante, estos son enviados hacia la plataforma utilizando un canal cifrado para la transferencia de los mismos. Además se utiliza el framework Hibernate, el que permite disminuir en gran medida los ataques por inyección SQL.

Portabilidad: El sistema se desarrolló utilizando Java – Apache - PostgreSQL, combinación que es compatible con los sistemas operativos más populares: Linux y *Windows*, solo se exige que dichos sistemas operativos tengan una implementación de la maquina virtual de Java.

Modificabilidad: El estilo de la arquitectura basada en componentes garantiza que se puedan realizar cambios en un componente y la adición de otros nuevos, sin afectar los ya existentes ni el funcionamiento de la plataforma, siempre y cuando mantengan correctas las interfaces de comunicación entre dichos componentes.

Fiabilidad: El sistema cuenta con un adecuado manejo de errores que permite una mayor estabilidad y recuperación ante cualquier anomalía, por lo cual el tiempo de recuperación en cada una de las fallas debe ser mínimo, si depende sólo de problemas de *software*.

Reusabilidad: El uso de arquitectura basada en componentes permite alcanzar un mayor nivel de reutilización de software, actualizar o agregar componentes sin afectar otras partes del sistema y ciclos de desarrollo más cortos para las nuevas aplicaciones. Es la forma más eficiente de reutilizar código y conocimiento ya existente, lo que conlleva a que esta pueda ser utilizada por cualquier operador de telefonía móvil en el mundo, sólo teniendo que realizar pequeños cambios.

Interoperatibilidad: La plataforma utiliza *WEB Services* para el intercambio de procesos o datos con agentes externos o distintas aplicaciones; aunque actualmente no se comunica con otros sistemas, los servicios *web* garantizan esta posibilidad.

Por el alto respeto del diseño arquitectónico con los requerimientos del sistema, se considera que la Arquitectura de la Plataforma de gestión de Contenidos para Celulares cumple cabalmente con las necesidades y expectativas del cliente.

4.7 Conclusiones

En este capítulo de describió la necesidad de evaluar la arquitectura de *software*, los momentos en que se debe evaluar, las personas que están involucradas en la evaluación, los parámetros por los que se puede evaluar, algunas técnicas y métodos de evaluación, así como, la valoración de la arquitectura del Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles.

La evaluación explícita de la arquitectura de los sistemas de *software* con respecto a los requerimientos de calidad, minimizará los riesgos de construir un sistema que falle y, consecuentemente, disminuirá el costo de desarrollo del sistema (50).

Conclusiones

Es de vital importancia que todo sistema de *software* esté respaldado por una arquitectura sólida que facilite su entendimiento, que sea capaz de organizar el desarrollo, fomentar la reutilización y hacer evolucionar el sistema.

Dado que la arquitectura de *software* es uno de los elementos fundamentales para la construcción de la Plataforma de Gestión de Contenidos para Dispositivos Móviles, se diseñó basada en componentes y, para ello, se realizó una investigación de varios estilos y patrones arquitectónicos que existen en la actualidad, para, de esta forma, hacer uso de las mejores técnicas.

Se presentaron varias vistas arquitectónicas entre las que figuran la de Casos de Uso, Lógica y Despliegue, para el mayor entendimiento de los miembros del proyecto; se definieron las herramientas y tecnologías necesarias para desarrollarlo y finalmente, se evaluó dicha arquitectura.

La evaluación de la arquitectura mostró el cumplimiento de los objetivos del presente trabajo, por medio de la alta correspondencia entre los requerimientos, los atributos de calidad y el diseño arquitectónico; ya que permitió la elaboración de una aplicación robusta, flexible y reusable.

La solución presentada ha permitido realizar una adecuada arquitectura de *software* para el Núcleo de la Plataforma de Gestión de Contenidos para Dispositivos Móviles.

Recomendaciones

- El refinamiento constante de la arquitectura.
- El análisis de las dependencias entre los subsistemas de implementación para evitar que existan cuellos de botella en alguno de ellos.
- Ir incorporando en el documento de la arquitectura, los diferentes patrones que se vayan utilizando a medida que se desarrollen otras versiones de la Plataforma.
- Crear un equipo encargado del estudio y aplicación de los métodos de evaluación de arquitectura de *software*, como parte del equipo de proyecto.
- Reutilizar la arquitectura de la primera versión del Núcleo de la Plataforma para la Gestión de Contenidos para Dispositivos Móviles en otros sistemas con características similares.

Referencias bibliográficas

1. **IEEE.** Recommended Practice for Architectural Description of Software-Intensive Systems. 2000. IEEE-Std-1471-2000.

2. **Pressman, R.** *Ingeniería de Software un Enfoque Práctico.* s.l. : McGraw-Hill.

3. **Dijkstra, Edsger.** *The Structure of the Multiprogramming system.* s.l. : Communications of the ACM, 1983.

4. **Brooks Jr, Frederick.** *The mythical man-month.* s.l. : Addison-Wesley, 1975.

5. **Shaw, M y Garlan, D.** *Software Architecture, Perspectives of an Emerging Discipline.* s.l. : Prentice Hall, 1996.

6. **Shaw, M y Clements, P.** *Documenting Software Architectures.* s.l. : Pearson, 2003.

7. **Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.** [En línea] [Citado el: 7 de abril de 2009.] http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/estiloypatron.pdf.

8. **SOA.** [En línea] [Citado el: 7 de abril de 2009.] http://www.accenture.com/NR/rdonlyres/4EDB1C67-F737-4FCD-B6DA-E14E665F1E0F/0/SOA.pdf.

9. **Aplicaciones Distribuidas.** [En línea] [Citado el: 18 de enero de 2009.] http://www.monografias.com/trabajos14/aplicacion-distrib/aplicacion-distrib.shtml.

10. **Velasco, D.** Un Lenguaje para la Especificación y Validación de Arquitecturas de Software. *Tesis doctoral.* s.l. : Universidad de Málaga, 2000.

11. **Jacobson, I y Rumbaugh, J.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison-Wesley Iberoamericana, 2000.

12. **XP.** Introducción a la Programación extrema. [En línea] [Citado el: 6 de mayo de 2009.] http://www.dsi.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf.

13. **Case.** Herramientas case. [En línea] [Citado el: 3 de mayo de 2009.] http://www.elprisma.com/apuntes/curso.asp?id=13324.

14. **Rational.** Rational Rose Enterprise. [En línea] [Citado el: 6 de mayo de 2009.] http://www.rational.com.ar/herramientas/roseenterprise.html.

15. **VP.** Visul Paradigm. [En línea] [Citado el: 10 de diciembre de 2008.] http://www.visual-paradigm.com.

16. **EA.** Enterprise Architect. [En línea] [Citado el: 6 de mayo de 2009.] http://www.sparxsystems.com.au/products/ea/index.html.

17. **SEI.** Architecture Description Languages. [En línea] [Citado el: 8 de abril de 2009.] http://www.sei.cmu.edu/architecture/adl.html.

18. **ACME.** Acme. [En línea] [Citado el: 31 de marzo de 2009.] http://www.cs.cmu.edu/acme/.

19. **ADL.** Arquitectura de software. [En línea] [Citado el: 31 de marzo de 2009.] http://siona.udea.edu.co/aoviedo/Arquitectura%20de%20Software/adl.htm.

20. **Jacal.** Jacal. [En línea] [Citado el: 31 de marzo de 2009.] http://siona.udea.edu.co/~aoviedo/Arquitectura%20de%20Software/adl.htm.

21. **Larman, Craig.** *UML y Patrones Introducción al análisis y diseño orientado a objetos.* s.l. : Prentice Hall, 1999.

22. **Java Sun.** Sun Microsystem. [En línea] [Citado el: 8 de abril de 2009.] http://java.sun.com/developer/technicalArticles/J2EE/jpa/&ei=zDLcSfOBB4_glQeehLHuDQ&sa=X&oi=translate&resnum=1&ct=result&prev=/search%3Fq%3DJPA%26hl%3Des%26sa%3DG.

23. **JCP.** SAMS. [En línea] [Citado el: 8 de abril de 2009.] http://jcp.org/en/jsr/detail?id=212.

24. **UCI.** Estándares de Codificación del proyecto Gina. [En línea] [Citado el: 21 de mayo de 2009.] http://10.31.18.2:5900/trac/ContentPlatformCode/attachment/wiki/WikiStart/Pautas.doc.

25. **Componentes.** Acerca de Desarrollo basado en componentes. [En línea] [Citado el: 6 de abril de 2009.] http://webcabcomponents.com/componentization.shtml.

26. **Componentes de software.** Qué son los componentes. [En línea] [Citado el: 6 de abril de 2009.] http://www.componentsource.com/services/about-us/components.html.

27. **Framework.** Frameworks. [En línea] [Citado el: 10 de marzo de 2009.] http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf.

28. **Walls, C y Breidenbach, R.** *Spring in Action.* s.l. : Manning Publications Co, 2005.

29. **Harrop, R y Machacek, J.** *Pro Spring.* s.l. : Apress, 2005.

30. **JUnit.** JUnit Framework. [En línea] [Citado el: 10 de diciembre de 2008.] http://www.junit.org.

31. **Hibernate.** Hibernate Framework. [En línea] [Citado el: 8 de abril de 2009.] http://www.hibernate.org/.

32. **Debian.** Debian: Operating System. [En línea] [Citado el: 18 de marzo de 2009.] http://www.debian.org/intro/about.

33. **DotProject.** DotProject. [En línea] [Citado el: 5 de febrero de 2009.] http://www.dotproject.net/.

34. **Trac.** Trac. [En línea] [Citado el: 18 de marzo de 2009.] http://trac.edgewall.org/.

35. **PBS.** Sistema Intregacion Continua. [En línea] [Citado el: 18 de febrero de 2009.] http://www.pbs.org/aboutpbs/.

36. **Sun.** Sun. [En línea] [Citado el: 9 de abril de 2009.] http://www.sun.com/.

37. **RapidSVN.** RapidSVN. [En línea] [Citado el: 5 de diciembre de 2008.] http://www.rapidsvn.org/faq.html.

38. **JEE.** Bloque de conocimientos de JEE. [En línea] [Citado el: 10 de enero de 2009.] http://www.epidataconsulting.com/tikiwiki/tiki-index.php?page=JEE#Desventajas.

39. **Eclipse.** Eclipse Ganymede. [En línea] [Citado el: 10 de enero de 2009.] http://www.eclipse.org/ganymede/.

40. **Plataforma eclipse**. Plataforma eclipse. [En línea] [Citado el: 11 de diciembre de 2008.] http://plataformaeclipse.com/.

41. **Alembik.** Alembik. [En línea] [Citado el: 21 de enero de 2009.] http://alembik.sourceforge.net/.

42. **Posgres.** PosgresSQL. [En línea] [Citado el: 21 de noviembre de 2008.] http://www.postgresql.org/docs/8.3/static/index.html.

43. **Apache.** Apache TomCat. [En línea] [Citado el: 20 de marzo de 2009.] http://tomcat.apache.org/tomcat-6.0-doc/index.html.

44. **Sosa, Mayret y Reina, Yoendris.** Módulo de Entrega de Contenido de la Plataforma de Gestión de Contenidos para Dispositivos Móviles. *Tesis de Pregrado, Universidad de las ciencias Informáticas.* La Habana : s.n., 2009.

45. **Rosquete, Raudel.** Módulo de Contenido de la Plataforma de Gestión de Contenido para Dispositivos Móviles. *Tesis de Pregrado, Universidad de las ciencias Informáticas.* La Habana : s.n., 2009.

46. **Echevarria, Enmanuel.** Módulo de Transcoder de la Plataforma de Gestión de Contenidos para Dispositivos Móviles. *Tesis de Pregrado, Universidad de las ciencias Informáticas.* La Habana : s.n., 2009.

47. **Ulacia, Yoandy.** Módulo de Pago de la Plataforma de Gestión de Contenidos para Dispositivos Móviles. *Tesis de Pregrado, Universidad de las ciencias Informáticas.* La Habana : s.n., 2009.

48. **Clements, P, Kazman, R y Klein, M.** *Evaluating Software Architectures: Methods and Case Studies.* s.l. : Addison Wesley, 2004.

49. **Quin, Z, Xing, J y Zheng, X.** *Software Architecture.* s.l. : Springer, 2008.

50. **Bosch, J.** *Design & Use of Software Architectures.* s.l. : Addison Wesley, 2000.

Glosario de términos

AOP: *(Aspect Oriented Programming)* Programación Orientada a Aspectos es un paradigma de programación que aumenta la modularidad, permitiendo la separación de las cuestiones transversales, formando una base para el desarrollo de *software* orientado a aspectos.

Casos de Uso: En ingeniería de *software*, es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de *software*. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico.

CDR: *(*Call Detail Record*) Registro que contiene información útil para la empresa sobre las transacciones realizadas por los clientes. Ej: día, fecha, hora, cliente, costo y contenido comprado, entre otros.*

CMS: *(Content Management System)* Sistema de gestión de contenidos es un programa que permite crear una estructura de soporte para la creación y administración de contenidos por parte de los participantes principalmente en páginas *web*. El sistema permite manejar de manera independiente el contenido y el diseño.

Componente: Más conocido como los componentes de *software*, que son todos aquellos recursos desarrollados para un fin concreto y que pueden formar solos o en conjunto con otros, un entorno funcional requerido por cualquier proceso predefinido. Son independientes entre ellos, y tienen su propia estructura e implementación. Si fueran propensos a la degradación debieran diseñarse con métodos internos propios de actualización.

Contenidos: Son los archivos que se pueden descargar de la Plataforma. Se clasifican como *Ringtones* e Imágenes, aunque dichas clasificaciones pueden tener subcategorías.

Directorio de Páginas Amarillas: Permite almacenar y consultar el número telefónico y la dirección de empresas, instituciones del estado, hospitales y otros servicios a la población.

Dispositivos Móviles: Para la realización de este trabajo se define como dispositivo móvil el conjunto formado por los teléfonos celulares y los PDA.

Gestión de Configuración: Conjunto de procesos destinados a asegurar la validez de todo producto obtenido durante cualquiera de las etapas del desarrollo de un sistema de información a través del estricto control de los cambios realizados sobre los mismos y de la disponibilidad constante de una versión estable de cada elemento para toda persona involucrada en el citado desarrollo.

Gestión de Proyecto: Busca las técnicas necesarias para planificar, organizar y supervisar proyectos de *software*. El objetivo de gestionar proyectos es tener un producto de alta calidad.

GoF: (*Gang of Four*) Banda de los cuatro, son 23 patrones de diseño orientado a objetos. La banda de los cuatro son generalmente considerados el fundamento de todos los demás patrones. Se clasifican en tres grupos: creacionales, estructurales y de comportamiento.

GPS: (*Global Positioning System*) Sistema de Posicionamiento Global es un sistema global de navegación por satélite que permite determinar en todo el mundo la posición de un objeto, una persona, un vehículo o una nave, con una alta precisión.

GRASP: (*General Responsibility Assignment Software Patterns*) Patrones generales de *software* para asignación de responsabilidades. Aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de *software*. Los nueve GRASP son: experto, creador, controlador, alta cohesión, bajo acoplamiento, polimorfismo, fabricación pura, indirecciones y no hables con extraños.

IDE: Por sus siglas en inglés *Integrate Development Environment*, 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. Pueden ser aplicaciones por sí solas, o pueden ser parte de aplicaciones existentes.

Integridad de la Información: Es la certeza de que el contenido de la información permanezca inalterado a menos que sea modificado por personal autorizado, y esta modificación sea registrada, asegurando su precisión y confiabilidad.

LOG: Registro que contiene información útil sobre las transacciones realizadas por los usuarios.

MMS: (*Multimedia Messaging Service*) es un servicio de mensajería que combina los mensajes de texto convencionales con otros tipos de contenido, como fotografías, imágenes, clips de sonido y vídeo.

PDA: (*Personal Digital Assistant*) Asistente Personal Digital, el cual se clasifica como una computadora de mano, sus prestaciones fundamentales son: navegación por internet, correo electrónico, juegos, música, películas, creación de documentos, calendarios, lista de contactos y recordatorios además de todas las funciones de los teléfonos celulares.

POJOs: *(Plain Old Java Object)* Viejos objetos planos de Java, son utilizados por programadores para enfatizar el uso de clases simples y que no dependen de un *framework* en especial. Surge como una reacción en el mundo Java a los *frameworks* cada vez más complejos, y que requieren un complicado andamiaje que esconde el problema que realmente se está modelando.

Promoción: Es una herramienta que consiste en incentivos de corto plazo, a los consumidores o clientes de ventas, que buscan incrementar la compra o la venta de un producto o servicio.

Proveedor*:* Es una empresa o persona que suministra contenidos o servicios a la empresa comercializadora, para posteriormente, brindarlos a los consumidores finales.

SAMS: (*Server API for Mobile Services*) establece una norma o mecanismo universal Java para aplicaciones de servidor para componer, enviar y recibir mensajes cortos (SMS) y mensajes multimedia (MMS). La API SAMS funciona independientemente de la correspondiente infraestructura de red y protocolos que se utilizan para la comunicación con el MMS *Center* y el SMS *Center Servers*.

SMS: (*Short Messagge Service*) Servicio de Mensajes Cortos. Mediante este servicio se pueden enviar mensajes de texto de hasta 160 caracteres a los móviles.

Teléfono Celular: El teléfono celular o móvil es un dispositivo inalámbrico electrónico que permite tener acceso a la red de telefonía. Su característica más importante es la portabilidad, que permite comunicarse desde casi cualquier lugar. Aunque su principal función es la comunicación de voz, como el teléfono convencional, su rápido desarrollo ha incorporado otras funciones como son mensajes cortos de texto y multimedia, cámara fotográfica, agenda, acceso a Internet, reproducción de video, música e incluso GPS.

URL: (*Uniform Resource Locator*) es un localizador uniforme de recurso. Es una secuencia de caracteres, de acuerdo a un formato estándar, que se usa para nombrar recursos, como documentos e imágenes en Internet, por su localización.

Vistas Arquitectónicas: Son descripciones simplificadas (abstracciones) de un sistema desde una perspectiva particular o punto de vista, que cubre particularidades y omite entidades que no son relevantes a esta perspectiva.

WAP: (*Wireless Applications Protocol*) Protocolo de Aplicaciones Inalámbricas. Es un protocolo basado en los estándares de Internet desarrollado para permitir a teléfonos celulares navegar a través de Internet.

WAP Push: Es un SMS que consiste en dos secciones, un mensaje de texto y un URL relacionado a una página WAP. Donde el teléfono móvil receptor del mensaje muestra el texto y brinda al usuario la opción de conectarse a la URL con sólo presionar una tecla.