UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

ESCUELA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES

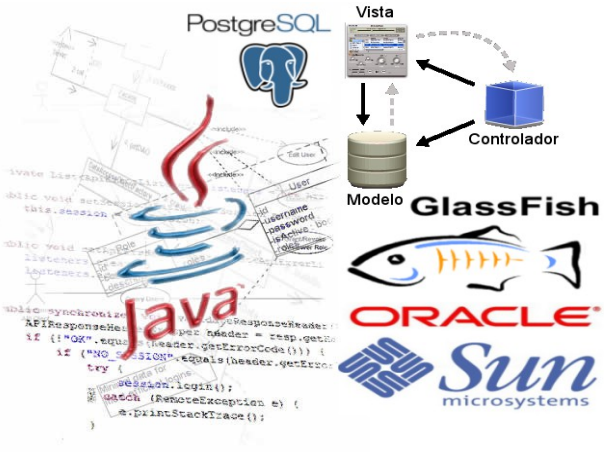
TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS COMPUTACIONALES

TEMA:

"ESTUDIO DE LA ARQUITECTURA DE SOFTWARE"

APLICATIVO:

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE PLANIFICACIÓN DE RECURSOS EMPRESARIALES PARA LA EMPRESA "FAUSTO DÍAZ" UTILIZANDO SOFTWARE LIBRE



AUTOR: EGDO. ALCIDES NEPTALÍ RIVERA POSSO

DIRECTOR: ING. IRVING REASCOS PAREDES

DICIEMBRE DE 2010

CERTIFICACIÓN

Certifico que el presente trabajo de investigación fue realizado por el Egdo. Alcides Neptalí Rivera Posso, bajo mi supervisión.

Atentamente,

Ing. Irving Reascos
DIRECTOR DE TESIS

DEDICATORIA

Dedico este trabajo a mis padres Fernando y Martha, a mis abuelitos Miguel e Inés, a mi hermano Miguel y a mi hermanita Fernanda por su apoyo incondicional para la culminación de mi carrera.

AGRADECIMIENTOS

Agradezco a mis padres, por todo su esfuerzo, consejos y dedicación puestos en mí.

Al Ing. Irving Reascos por sus enseñanzas y su ayuda en mi trabajo final de grado.

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

CAPÍTULO I

1 ARQUITECTURA DE SOFTWARE

1.1 INTRODUCCIÓN

En los inicios de la informática, la programación se consideraba un arte y se desarrollaba como tal, debido a la dificultad que entrañaba para la mayoría de personas, pero con el tiempo se ha ido descubriendo y desarrollando formas y guías generales, con base a las cuales se pueden resolver los problemas.

A estas, se las han denominado Arquitectura de Software, porque, semejanza de los planos de un edificio o construcción, estas indican la estructura, funcionamiento e interacción entre las partes de software.

En su libro *"An Introduction to Software Architecture"* David Garlan y Mary Shaw definen que la "*Arquitectura es un nivel de diseño que hace foco en aspectos mas allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un tipo de problema."*

Se utiliza el término *"Arquitectura"* en contraste con *"Diseño"*, para evocar nociones de codificación, de abstracción, de estándares de entrenamiento formal (de los arquitectos de software) y de estilo. Es tiempo de re – examinar el papel de la arquitectura de software en el contexto más amplio del proceso de software y de su administración, así como señalar las nuevas técnicas que han sido adoptadas.

A pesar de que hoy en día el intercambio de información resulta más sencillo; la industria y la academia muchas veces no definen las mismas prioridades, diagnostican las situaciones de manera distinta, y utilizan las mismas nomenclaturas sin compartir sus significados.

Como lo ha dicho Jan Bosch[1], un arquitecto práctico: *"Existe una considerable diferencia entre la percepción académica de la Arquitectura de Software y la práctica industrial… Es interesante advertir que a veces los problemas que la industria identifica como los más importantes y difíciles, no se identifican o se consideran no – problemas en la academia"* [LIB01].

El desarrollo de software se ha vuelto un proceso largo y muy complejo, en donde el problema más importante ya no radica en los algoritmos, ni en las estructuras de datos a utilizar, sino en la toma de decisiones organizacionales.

Existe entonces el espacio oportuno para comenzar a entender todo el proceso de desarrollo de software en el marco de las tendencias actuales de la teoría y práctica arquitectónica. Lo que nos ayudará a correlacionar la investigación básica y los aportes académicos con las visiones y requerimientos de la industria.

1 **Jan Bosch,** Ingeniero de Software, profesor, consultor y ejecutivo; es uno de los arquitectos de software más reconocidos a nivel mundial [WWW01].

1.2 DEFINICIONES

Existen múltiples definiciones acerca de la Arquitectura de Software entre las que tenemos:

Una definición reconocida es la de Clements[2]: *"La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones"* [LIB02].

Otra definición según el documento de la IEEE[3] Std. 1471 – 2000, adoptada también por Microsoft dice: *"La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orienten su diseño y evolución".*

2 **Clements,** Antes de llegar al SEI (Instituto de Ingeniería de Software) en 1994, el Dr. Paul Clements trabajo para el Naval Resarch Laboratory de los EE. UU. Allí participó en el proyecto SCR (Reducción de Costos del Software) o "A-7". El SCR es pionera en técnicas de diseño de software modular, los requisitos y especificaciones de ingeniería, arquitectura de software y estructuras arquitectónicas, especificación de interfaz y documentación, y en tiempo real el rendimiento de la ingeniería [WWW02].

3 **IEEE,** corresponde a las siglas de *The Institute of Electrical and Electronics Enginners,* el Instituto de Ingenieros Eléctricos y Electronicos, una asociación técnico – profesional mundial dedicada a la estandarización, entre otras cosas [WWW03].

Según Kruchten Philippe[4]: *"La Arquitectura de Software, tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad y disponibilidad."*

Lane[5] define la Arquitectura de Software como: *"… Arquitectura de Software es el estudio de la estructura a gran escala y el rendimiento de los sistemas de software. Aspectos importantes de la arquitectura de un sistema incluye la división de funciones entre los módulos del sistema, los medios de comunicación entre los módulos, y la representación de la información compartida"* [WWW06].

De acuerdo a las definiciones antes mencionadas podemos decir entonces que la Arquitectura de Software no es más que tener una vista global y abstracta del sistema que se desea desarrollar, que nos permita tomar las decisiones adecuadas, definir los parámetros y delimitación de dicho sistema tomando en cuenta los requerimientos funcionales y no funcionales.

4 **Kruchten Philippe,** es un ingeniero canadiense de software, y profesor de Ingeniería de Software en la Universidad Columbia Británica en Vancouver de Canadá, conocido como el Director del Proceso de Desarrollo (RUP) en Rational Software, y desarrollador del modelo 4+1 vistas [WWW04].

5 **Jim (James) Lane,** originario del norte de New Jersey, miembro del staff original de Microsoft, trabajó en la creación de la relación entre Microsoft e Intel, la llamada alianza "Wintel". Lane trabajó en el compilador de Pascal de Microsoft, y fue jefe de proyecto del compilador de FORTRAN antes de abondonar Microsoft en 1985, desde entonces posee una pequeña empresa de desarrollo de software [WWW05].

1.3 UTILIZACIÓN

La Arquitectura de Software comprende una cantidad de toma de decisiones acerca de la organización de un sistema: la selección de los elementos estructurales y las interfaces por las cuales un sistema está compuesto, junto con su comportamiento que es especificado por la colaboración entre estos elementos [WWW07].

Por lo tanto utilizando la Arquitectura de Software se logrará un software que tendrá las siguientes características:

Portabilidad: El software será apto de ejecutarse en cualquier plataforma independientemente de su hardware y software de base, configurando tanto aplicaciones para intranet como para Internet.

Base de Datos: Las base de datos serán relacionales, objeto – relacionales, orientadas a objetos y podrán estar local o remotamente.

Escalabilidad: Se podrán incrementar los módulos sin necesidad de modificar el código de base.

Seguridad: Se proporcionará las seguridades necesarias para el software asignando los roles adecuados a cada uno de los usuarios.

Interfaz de Usuario: Podremos contar con aplicaciones web o inclusive podrán ser entorno de ventanas.

1.4 EL ARQUITECTO DE SOFTWARE

Ilustración 1: *El Arquitecto (Tomado de la saga de películas "The Matrix")*

El término Arquitecto de Software se ha convertido en el título de moda en toda empresa de sistemas o con una área propia de sistemas. Decimos de moda, debido a que no todas las empresas necesitan realmente arquitectos de software, y tal vez, ni siquiera todos los proyectos necesiten de un verdadero arquitecto de software.

Es común que muchas tareas relevantes de un proyecto puedan ser resueltas con un desarrollador experimentado, sin tener la necesidad de contratar un arquitecto. Muy frecuentemente se tiende a confundir estos dos perfiles, que son abismalmente diferentes.

También es importante notar la diferencia entre los *"gurúes tecnológicos"* y los verdaderos arquitectos. Estas cuestiones aumentan la confusión existente sobre que es un arquitecto y cuáles se supone tendrían que ser sus responsabilidades [WWW09].

1.4.1 ROL DE LOS ARQUITECTOS

El rol de los arquitectos suele comprender las siguientes tareas:

- Definición de las vistas de la arquitectura de una aplicación (o sea, *"crear"* la arquitectura, ya que la arquitectura, en pocas palabras es un conjunto de vistas de alto nivel).
- Dar soporte técnico – tecnológico a desarrolladores, clientes y expertos en negocios.
- Conceptualizar y experimentar con distintos enfoques arquitectónicos.
- Crear documentos de modelos, componentes y especificaciones de interfaces.
- Validar la arquitectura contra requerimientos, suposiciones; y además.
- Tener una dosis de estrategia y política, es decir, ser en parte un consultor.

De esta manera logramos una figura que probablemente se ajuste a cualquier realidad (adaptando algunos puntos específicos de sus tareas) [WWW09].

Según Rational Unified Process[6] Arquitecto es un rol en un proyecto de desarrollo de software el cual es responsable de:

- Liderar el proceso de arquitectura.
- Producir los artefactos necesarios: Documento de descripción de arquitectura.
- Modelos y prototipos de arquitectura.

6 **Información obtenida de:** Philippe Krutchen, *"The Rational Unified Process",* Pearson Education, 1999.

Según Sun Microsystems[7] el arquitecto:

- Visualiza el comportamiento del sistema.
- Crea los planos del sistema.
- Define la forma en la cual los elementos del sistema trabajan en conjunto.
- Responsable de integrar los requerimientos no – funcionales en el sistema.

1.4.1.1 RIESGOS Y RECOMPENSAS DEL ROL DE LOS ARQUITECTOS

RIESGOS:

- Tu no disfrutas del trabajo no técnico.
- Más responsabilidades con menos control.
- Dirección insuficiente para la resistencia al cambio.
- Pocas posibilidades de éxito.
- Todo el mundo tiene un mejor idea.

RECOMPENSAS:

- Problemas más interesantes y complejos.
- Reconocimiento y promoción profesional.
- Mayor contribución y alcance de tus actividades.

7 **Información obtenida de:** Sun, *Developing Architectures for Enterprise Java Applications (SL-425)* http://www.sun.com/training/catalog/courses/SL-425.xml

1.4.2 DOMINIO DE LOS ARQUITECTOS

En el quehacer cotidiano del arquitecto de software, existen varias tareas o dominios (independientemente de sus propias tareas incluidas en el ciclo de vida del proyecto) en los que suelen estar enfocado, estos son:

Tecnología: Es encargado de revisar las posibles alternativas de solución a través de distintos modelos, preparar los documentos técnicos y; convencer y comunicar de la factibilidad de los proyectos a los sponsors[8] y stakeholders[9].

Estrategias de negocio: Tener conocimiento acerca de los procesos internos y externos de la empresa, para saber como *"vender"* pero no desde el punto de vista comercial sino de cómo nuestras herramientas nos ayuden al crecimiento comercial de la empresa.

Comunicación: Políticas de organización, conocer a los stakeholders sus necesidades y requerimientos; tener una comunicación fluida con ellos. Encargado de la generación de reportes y la comunicación de resultados.

Liderazgo: En todo y ante todo un arquitecto de software debe tener liderazgo, es quién colaborará directamente con el jefe del proyecto en todo el proceso de desarrollo; y es el puente perfecto entre desarrolladores y los expertos del negocio.

8 **Sponsors,** ejecutivos de la gerencia superior que apadrinan la iniciativa de mejoramiento y que tienen la capacidad de proveer los recursos necesarios para su ejecución.

9 **Stakeholders,** es aquella persona o entidad que esta interesada en la realización de un proyecto o tarea, auspiciando el mismo ya sea mediante su poder de decisión o de financiamiento [WWW10].

Además los arquitectos también forman parte del desarrollo de un proyecto. Durante este proceso, las fases más comúnmente reconocidas como más importantes del Arquitecto de Software son:

Pre – diseño: Entender el alcance del proyecto, los puntos más importantes del diseño: validar y manejar requerimientos; y expectativas del cliente.

Análisis del dominio: Entender en detalle los requerimientos del cliente, crear bocetos de los comportamientos deseados del sistema.

Diseño esquemático: Definir el Look & Feel[10], si es necesario construir prototipos.

Desarrollo del diseño: Ampliar los detalles y refinar el diseño para llegar al diseño final.

Documentos del proyecto: Soporte a desarrolladores: lidiar con problemas concretos, revisión de código para controlar la calidad, ver cómo funcionan (o no) las cosas.

Definir el proceso de desarrollo: los roles de los miembros del equipo y la secuencia de construcción de la aplicación, especificar metodologías y tecnologías; y definir todos los detalles necesarios para todos aquellos que construirán la aplicación.

10 **Look & Feel,** este término inglés hace referencia a los diferentes aspectos y funcionamiento de las interfaces gráficas de usuario (GUI). Así, una interface basada en ventanas como Windows tiene un mejor "Look & Feel" que una basada en menús desplegables o bajo DOS, por ejemplo. Un "Look & Feel" (Parecer y Percepción) excelente, puede representar la diferencia entre un portal exitoso y un portal sin tráfico, en fin el "Look & Feel", es lo que se ve y se percibe al navegar por un portal de internet [WWW11].

Contratación o staffing: Seleccionar desarrolladores; si el desarrollo es tercerizado, participar en la elección del proveedor.

Construcción: Asegurar que la visión del cliente sea mantenida y respetada durante el desarrollo. Revisar y validar los diseños del nivel de construcción si la complejidad de los mismos lo amerita, diseñar modificaciones pedidas por el cliente y participar en el testeo y aceptación de revisiones solicitadas por el cliente.

Post – construcción: Asistencia en la migración del sistema nuevo (implementación) puede llegar a manejar el entrenamiento de los usuarios nuevos y definir los procesos de mantenimiento.

1.5 EL ICEBERG DE LA USABILIDAD

Hasta hace poco, se asumía que la usabilidad era una propiedad exclusiva de la presentación de la información. Se creía que, encapsulando la capa de presentación y separándole del resto, se podía desarrollar la aplicación y, de forma iterativa, pasar los test de usabilidad.

Tras cada test, tan sólo será necesario resolver los problemas modificando la presentación y, gracias a esta separación, la funcionalidad no quedaría afectada.

En realidad, este modelo de desarrollo ha fallado ha menudo. ¿Cuántas veces hemos tenido que correr a realizar cambios profundos en la funcionalidad de una aplicación después de haber detectado problemas de usabilidad? Dick Berry[11], en su analogía del Iceberg de la Usabilidad, explica que los aspectos relacionados con la presentación, es decir, lo que normalmente entendemos como Look & Feel, sólo afectan en un 40% a la usabilidad.

El 60% restante está influenciado por lo que él llama *"modelo de usuario"*, que esta constituido por los objetivos que el usuario quiera alcanzar con sus tareas, tal como se muestra en la ilustración número 2[12]. [WWW12].

11 **Dick Berry,** es un distinguido ingeniero de IBM, miembro del equipo de Diseño y Arquitectura en Austin Texas. Su enfoque desde 1980 ha sido el diseño de intefaces humano – computador, modelos de objetos de usuario, y metodologías para facilitar el uso del diseño y desarrollo. Fue el principal arquitecto de varias generaciones del Acceso Común de Usuario (CUA), una interfaz de usuario incialmente publicada en 1987 [WWW13].

12 **Ilustración tomada de:** http://www.desarrolloweb.com/articulos/1622.php

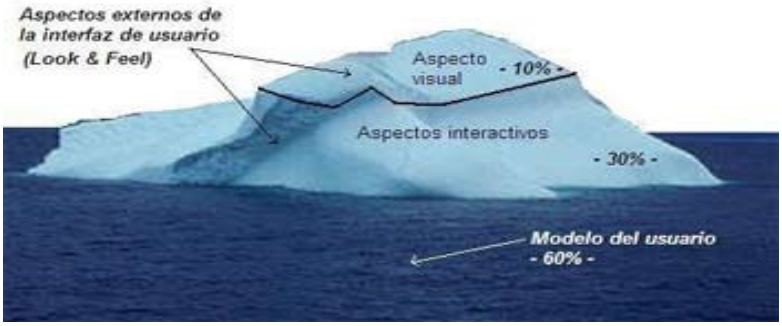


Ilustración 2: *Analogía del Iceberg de Usabilidad*

1.5.1 USABILIDAD Y ARQUITECTURA DE SOFTWARE

Se debe tener en cuenta la usabilidad desde el inicio del proyecto de desarrollo de software, es decir, lo que se denomina momento de Arquitectura de Software.

Sabemos además, que cuando más tarde se detectan los problemas más cuesta arreglarlos, cuántas veces nos ha ocurrido que cuando estamos diseñando la interfaz de un nuevo sistema queremos crear diálogos e interacciones que el entorno tecnológico no nos permite, y terminamos exclamando ¡pero si esto ya lo he hecho en otra aplicación!.

Si analizamos estos escenarios de interacción, veremos que la causa de que no se puedan implementar es que no se tuvo en cuenta al usuario al inicio del diseño del sistema, es decir, en la Arquitectura del Software [WWW12].

1.6 LENGUAJES DE DESCRIPCIÓN ARQUITECTÓNICA

El arquitecto de software, luego de conocer los requerimientos, se propone delinear su estrategia y articular los patrones que se ofrecen hoy en abundancia, se supone que debería expresar las características del sistema, o en otras palabras, modelarlo, aplicando una convención gráfica o algún lenguaje avanzado de alto nivel de abstracción [WWW14].

Debido a la gran importancia de la descripción arquitectónica, y a su uso como medio de comunicación entre los diferentes stakeholders, se han tratado de desarrollar diferentes lenguajes para soportar esta descripción, algunos han tenido éxito, y otros simplemente han fracasado [WWW15].

1.6.1 DEFINICIONES

Un Lenguaje de Descripción Arquitectónica (ADL[13]) es un lenguaje descriptivo de modelado que se focaliza en la estructura de alto nivel de la aplicación antes que en los detalles de implementación de sus módulos concretos [LIB03].

Un ADL debe modelar explícitamente los componentes, los conectores y las diversas configuraciones; más aún, para que sea realmente utilizable y útil debe facilitar un soporte de herramienta para el desarrollo arquitectónico.

Por lo tanto, un ADL es un lenguaje que proporciona características para modelar la arquitectura conceptual de un sistema software, distintiva de la implementación del sistema.

13 **A**rchitecture **D**escription **L**anguage, son las siglas en inglés de un Lenguaje de Descripción Arquitectónica

Los ADL's proporcionan tanto una sintaxis concreta como un marco conceptual para la caracterización de arquitecturas donde esté refleje las características del dominio para los que el ADL está enfocado y/o el estilo arquitectónico.

1.6.2 CRITERIOS DE DEFINICIÓN DE UN ADL

Los criterios que vamos a tomar en cuenta son los que se encuentran en Vestal [LIB03], quien sostiene que un ADL debe modelar o soportar los siguientes conceptos:

- Componentes.
- Conexiones.
- Composición jerárquica, en la que un componente puede contener una subarquitectura completa.
- Paradigmas de computación, es decir, semánticas, restricciones y propiedades no funcionales.
- Paradigmas de comunicación.
- Modelos formales y subadyacentes.
- Soporte de herramientas para modelado, análisis, evaluación y verificación.
- Composición automática de código aplicativo.

1.6.3 CARACTERÍSTICAS GENERALES DE LOS ADL's

- Una sintaxis gráfica asociada a la sintaxis textual y, en cualquier caso, encontraremos una semántica bien definida.

- Un enfoque hacia el modelado de sistemas distribuidos.

- Un escaso apoyo en la captura de la *"rationale"*[14] del diseño y/o su historia: quizás sólo llegan a tener algunos mecanismos de notación de propósito general.

- Una gestión de los flujos de datos y de control como mecanismos de interconexión mientras que otras clases de conexión más específicas están menos toleradas.

- Una capacidad para representar diversos niveles jerárquicos de detalle y gestionar las distintas vistas de una arquitectura.

Sin embargo, existen muchas otras características que difieren ampliamente de un ADL a otro, entre las que encontramos:

- La capacidad para gestionar constructores de tiempo real a nivel arquitectónico.

- La capacidad para respaldar la especificación de estilos arquitectónicos particulares.

- La capacidad de personalización (casi nula en todos ellos) de las reglas de consistencia y completitud que incorporan.

- La capacidad para realizar un análisis de validez.

- La capacidad para gestionar la variabilidad o instancias diferentes de una misma arquitectura: no soportan las arquitecturas de línea de productos.

14 **Rationale**, Rational Unificad Process (RUP), es una metodología de desarrollo de software propuesta por IBM, para lo cual proporciona un grupo de herramientas.

1.6.4 LENGUAJES

A continuación una tabla que muestra los principales lenguajes de descripción arquitectónicas.

ADL	FECHA	OBSERVACIONES
Acme	1995	Lenguaje de intercambio de ADL's
Aesop	1994	ADL de propósito general, énfasis en los estilos
ArTek	1994	Lenguaje específico de domino.
Armani	1998	ADL asociado a Acme.
C2 SADL	1996	ADL especifico de dominio
CHAM	1990	Lenguaje de especificación.
Darwin	1991	ADL con énfasis en dinámica.
Jacal	1997	ADL – notación de alto nivel para descripción y prototipo.
LILEANNA	1993	Lenguaje de conexión de módulos.
MetaH	1993	ADL especifico de dominio.
Rapide	1990	ADL & simulación.
SADL	1995	ADL con énfasis en mapeo de refinamiento.
UML	1995	Lenguaje genérico de modelado.
UniCon	1995	ADL de propósito general, énfasis en conectores y estilos.
Wright	1994	ADL de propósito general, énfasis en comunicación
xADL	2000	ADL basado en XML.

Tabla 1: *Listado de los principales ADL's*

Ninguno se impuso ni en la academia ni en el mercado; existe un compás con los nuevos ADL's que se encuentran en el mercado como son: UML 2.0, Lenguajes Específicos de Dominio (Domain Specific Languages), XML y Web Semántica.

1.7 VISTAS ARQUITECTÓNICAS

Las vistas arquitectónicas representan un aspecto parcial de una arquitectura de software que muestran propiedades específicas de los sistemas.

La arquitectura de un sistema consta de múltiples vistas, asociadas a diferentes dimensiones o perspectivas del sistema, ninguna vista en particular constituye la arquitectura del sistema.

Las vistas se encuentran dirigidas a usuarios particulares y asociadas a requisitos no – funcionales concretos [WWW16].

A continuación se muestran ciertos elementos que forman parte de las vistas arquitectónicas:

- Punto de vista de los involucrados y puntos de vistas de los mismos.
- Elementos que serán capturados y representados en la vista y las relaciones entre los mismos.
- Principios para organizar la vista.
- Forma en que se relacionan los elementos de una vista con otra vista.
- Proceso a ser utilizado para la creación de la vista.

1.7.1 VISTA ARQUITECTÓNICA DE JOHN ZACHMAN

Consiste en una matriz de seis filas y seis columnas, compuesto por 36 celdas.

Tiene seis niveles de arquitectura que se desarrollan en filas: ámbito, modelo de empresa, modelo de sistema, modelo tecnológico, detalles de la representación y sistemas funcionales. Los tres primeros niveles son conceptuales y los siguientes son detalles del diseño y la construcción de los sistemas.

En columnas se representan las diferentes áreas de interés: datos, procesos, redes, personas, tiempo y motivación. Se corresponden con el qué, cómo, dónde, quiénes, cuándo y por qué.

Por lo tanto, se tiene una especificación formal para cada área de interés y cada nivel. La idea es ofrecer la información necesaria, uniformemente comprendida a todos los niveles, utilizando representaciones gráficas o lenguaje escrito [LIB04].

	DATOS (QUÉ)	**PROCESOS (CÓMO)**	**REDES (DONDE)**	**PERSONAS (QUIÉNES)**	**TIEMPO (CUÁNDO)**	**MOTIVACIÓN (POR QUÉ)**
ÁMBITO (CONTEXTO)	Lista de cosas importantes del negocio	Lista de Procesos que ejecutan el negocio	Lista de lugares en donde opera el negocio	Lista de Organizaciones importantes para el negocio	Lista de eventos significativos del negocio	Lista de metas estratégicas de negocio
MODELO DE EMPRESA (CONCEPTUAL)	Semántica del modelo	Modelo de Procesos de negocio	Sistema logístico del negocio	Modelo del Flujo de trabajo	Programación maestra	Plan de negocios
MODELO DE SISTEMA (LÓGICO)	Modelo Lógico de Datos	Arquitectura de aplicaciones	Arquitectura distribuida de sistemas	Arquitectura de interfaces humanas	Estructura de procesamiento	Modelo de reglas del negocio
MODELO TECNOLÓGICO (FÍSICO)	Modelo Físico de Datos	Diseño del sistema	Arquitectura tecnológica	Arquitectura de representaciones	Estructura de control	Diseño de reglas
DETALLES DE LA PRESENTACIÓN (FUERA DE CONTEXTO)	Definiciones de Datos	Programas	Arquitectura de redes	Arquitectura de seguridad	Definición de *timing*	Especificación de reglas
SISTEMAS FUNCIONALES						

Tabla 2: *Matriz de Zachman*

No obstante, hay que reconocer que tres de las vistas propuestas por Zachman (conceptual, lógica y física) corresponden a los marcos de referencia para vistas arquitectónicas posteriores.

1.7.2 VISTA ARQUITECTÓNICA DE PHILIPPE KRUCHTEN

Philippe Kruchten propuso el modelo “4+1”, vinculado al Rational Unified Process (RUP), que define cuatro vistas diferentes de la arquitectura de software [LIB05]:

Vista lógica: Describe el modelo de objetos.

Vista de proceso: Muestra la concurrencia y sincronía de los procesos.

Vista física: Muestra la ubicación del software en el hardware.

Vista de desarrollo: Describe la organización del entorno de desarrollo.

Existe una quinta vista que consiste en una selección de casos de uso o de escenarios que los arquitectos puedan elaborar a través de las vistas anteriores.

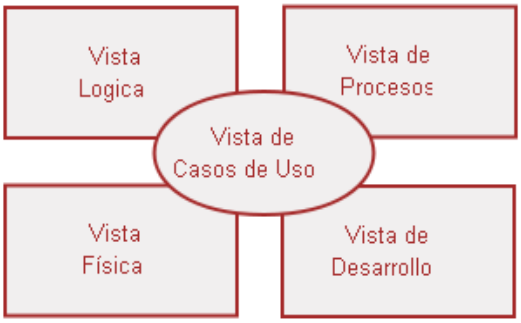


Ilustración 3: *Vista arquitectónica de Kruchten*

La ilustración número 3[15] muestra la vista arquitectónica de Philippe Kruchten este modelo "4+1" se percibe con un intento de reformular una arquitectura estructural o descriptiva en términos de objeto y de UML.

1.7.3 VISTA ARQUITECTÓNICA DE GRADY BOOCH, JAMES RUMBAUGH E IVAR JACOBSON

Consta de un esquema de cinco vistas las cuales muestran información diferente sobre el sistema [LIB06]. La ilustración número 4[16] muestra la vista arquitectónica propuesta por Grady Booch, James Rumbaugh e Ivar Jacobson en su introducción a UML.

15 **Ilustración tomada de:** Philippe Kruchten, The 4+1 View Model of Architecture, IEEE Software, 1995.

16 **Ilustración tomada de:** Grady Booch, James Rumbaugh e Ivar Jacobson, El Lenguaje Unificado de Modelado, Addison-Wesley, 1999.

Ilustración 4: *Vista arquitectónica UML*

Vista de Casos de Uso: Muestra información sobre el comportamiento y funcionalidad del sistema.

Vista de Diseño: Proporciona información del vocabulario y la funcionalidad del sistema.

Vista de Interacción: Nos da información sobre el rendimiento del sistema, la escalabilidad del mismo y la capacidad de procesamiento necesario.

Vista de Implementación: Establece el ensamblado del sistema y la gestión de la configuración.

Vista de Despliegue: Permite establecer la topología del sistema, su distribución y las pautas para su instalación.

1.7.4 VISTA ARQUITECTÓNICA DE BASS, CLEMENTS Y KAZMAN

Bass, Clements y Kazman presentan en 1998 un modelo que consta de nueve vistas, orientadas hacia el diseño concreto y la implementación [LIB07]:

Estructura de módulo: Las unidades son asignaciones de tarea.

Estructura lógica o conceptual: Las unidades son abstracciones de los requerimientos funcionales del sistema.

Estructura de procesos o de coordinación: Las unidades son procesos o threads.

Estructura física: Las unidades son los diferentes equipos de comunicación y servicios.

Estructura de uso: Las unidades son procedimientos o módulos, vinculados por relaciones de presunción de presencia correcta.

Estructura de llamados: Las unidades son usualmente subprocedimientos, vinculados por invocaciones o llamados.

Flujos de datos: Las unidades son programas o módulos, las relación es de envío de datos.

Flujo de control: Las unidades son programas, módulos o estados del sistema.

Estructura de clases: Las unidades son objetos.

1.7.5 COMPARACIÓN DE VISTAS ARQUITECTÓNICAS

A continuación realizaremos una comparación entre las diferentes vistas, para lo cual se mencionan las siguientes semejanzas y diferencias entre las diferentes perspectivas:

PERSPECTIVA	ZACHMAN	KRUCHTEN	BOOCH, RUMBAUGH Y JACOBSON	BASS, CLEMENTS Y KAZMAN	USUARIOS	ATRIBUTO DE CALIDAD
Abstracción de requerimientos funcionales del sistema	Vista Funcional	Vista Lógica		Vista Conceptual o Lógica	Cliente Usuario final Analista	Modificabilidad Reusabilidad Dependencia Seguridad Externa
Creación de procesos e hilos de ejecución, comunicación entre ellos y recursos compartidos	Vista de Concurrencia	Vista de Proceso	Vista de Despliegue	Vista de procesos o coordinación + Vista de llamadas	Arquitectos Desarrolladores Equipo de Pruebas Mantenimiento	Desempeño Disponibilidad
Organización de los elementos arquitectónicos implementados	Vista de Desarrollo	Vista de Implantación	Vista de Implementación	Vista Física + Vista de Módulos	Programadores Mantenimiento Gerentes de Configuración Gerentes de Desarrollo	Mantenibilidad Modificabilidad Capacidad de Prueba
Distribución de procesos en la plataforma	Vista Física + Vista de Concurrencias	Vista de Desarrollo	Vista de Interacción	Vista de Flujo de Control	Arquitectos Desarrolladores Equipo de Pruebas Mantenimiento Ing. Hardware	Desempeño Escalabilidad Disponibilidad Seguridad Interna
Escenarios y casos de uso		Vista de Casos de Uso	Vista de Casos de Uso	Vista de Usos	Cliente Usuario final Analista	Reusabilidad Disponibilidad Modificabilidad
Especificación abstracta de clases, objetos, funciones y procedimientos	Vista de Código		Vista de Diseño	Vista de Clases + Flujo de Datos	Diseñadores Desarrolladores	Modificabilidad Portabilidad Mantenibilidad

Tabla 3: *Comparación de vistas arquitectónicas en función de las perspectivas del sistema*

Así como las vistas arquitectónicas ofrecen una descripción de una arquitectura, existen notaciones que, mediante un conjunto definido de elementos y formas de representación, permiten de igual manera establecer la descripción de una arquitectura de software. Este es el caso del Lenguaje de Modelado Unificado (UML).

Cada paradigma de desarrollo exige diferente número y tipo de vistas y modelos para describir una arquitectura. No obstante, existen al menos tres vistas absolutamente fundamentales en cualquier arquitectura:

Visión estática: *D*escribe los componentes que posee una arquitectura.

Visión funcional: *D*escribe las funciones de cada componente.

Visión dinámica: Describe la forma en que se comportan los componentes a lo largo del tiempo y la manera que interactúan entre sí.

Las vistas de una arquitectura pueden expresarse mediante uno o varios lenguajes. El más obvio es el lenguaje natural, pero existen otros lenguajes tales como los diagramas de estado, los diagramas de flujo de datos, etc.

Estos lenguajes son apropiados únicamente para la vista, existe un cierto consenso en adoptar UML como lenguaje único para los modelos o vistas. Sin embargo, adoptar un sólo lenguaje puede correr el riesgo de no ser capaz de describir la información de un sistema en particular.

1.8 ESTILOS ARQUITECTÓNICOS

"Los Estilos Arquitectónicos son un conjunto de reglas de diseño que identifican las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo." [LIB08]

Un estilo arquitectónico es una descripción del patrón de los datos y la interacción de control entre los componentes, ligada a una descripción informal de los beneficios e inconvenientes aparejados por el uso del estilo. Los estilos arquitectónicos son artefactos de ingeniería importantes por que definen clases de diseño junto con las propiedades conocidas asociadas a ellos [LIB09].

Entre los principales estilos arquitectónicos tenemos:

- Arquitecturas centradas en datos.
- Arquitecturas de flujo de datos.
- Arquitecturas de llamada y retorno.
- Arquitecturas orientadas a objetos.
- Arquitecturas orientadas a servicios.
- Arquitecturas orientadas a aspectos.
- Arquitecturas estratificadas.

Un problema puede satisfacerse mediante estructuras a las que se llegarán posiblemente utilizando técnicas distintas. A veces la diferencia entre dos estilos no ésta muy clara, lo que provoca que haya mezclas entre ellos.

1.8.1 ARQUITECTURAS CENTRADAS EN DATOS

En la parte central de esta arquitectura se encuentra un almacén de datos (por ejemplo, un documento o base de datos) al que otros componentes acceden con frecuencia para actualizar, añadir o borrar los datos del almacén [LIB10].

Entre los repositorios de las Arquitecturas Centradas de Datos tenemos:

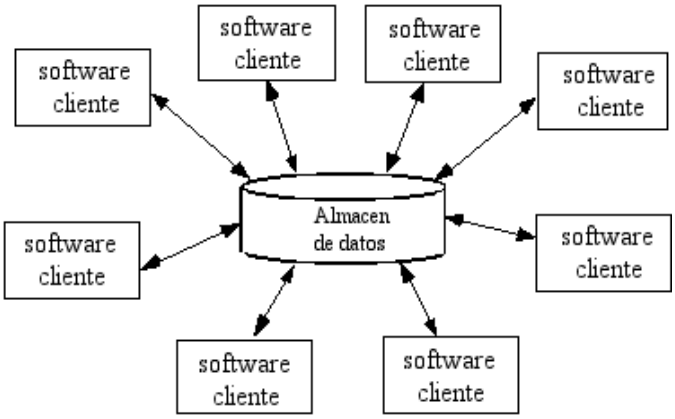
Repositorio pasivo: El *Software Cliente* accede a los datos independiente de cualquier cambio en los datos o a las acciones de otros *Software Cliente*.

Repositorio activo (pizarra): El repositorio envía información a los *Software Cliente* cuando los datos de su interés cambian, siendo por tanto un ente activo.

Las Arquitecturas Centradas de Datos brindan integridad, es decir pueden existir cambios en los componentes sin que esto afecte a los otros *Software Cliente*, además de mecanismos de transferencia de información entre los *Software Cliente*.

Algunos ejemplos:

- Motores de búsqueda.
- Bibliotecas de componentes reutilizables.
- Grandes bases de datos.



La ilustración número 5[17] muestra un ejemplo del estilo de arquitectura centrado en datos.

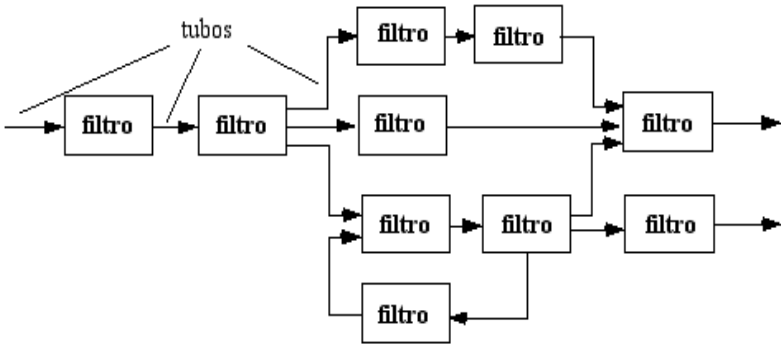
1.8.2 ARQUITECTURAS DE FLUJO DE DATOS

Arquitectura aplicada cuando los datos de entrada son transformados a través de una serie de componentes computacionales en los datos de salida [LIB10].

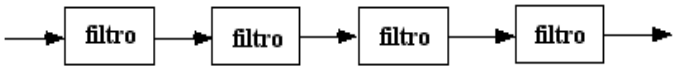
Se basa prácticamente en el patrón *"pipe and filter"* (tuberías y filtros). Este patrón se encuentra constituido de un conjunto de componentes denominados "filtros" conectados entre sí por "tuberías" que transmiten los datos desde un componente hacia otro.

Los filtros se encuentran diseñados para recibir la entrada de datos de una forma y producir la salida de datos en una forma específica. Si el flujo de datos genera una línea de transformaciones se denomina *Secuencial por Lotes* [WWW20].

17 **Ilustración tomada de:** Roger Pressman, Ingeniería del Software: Un enfoque práctico, McGraw Hill, 2001.



(a) tubos y filtros



(b) Secuencial por lotes

La ilustración número 6[18] muestra un ejemplo del estilo de arquitectura de flujo de datos.

1.8.3 ARQUITECTURAS DE LLAMADA Y RETORNO

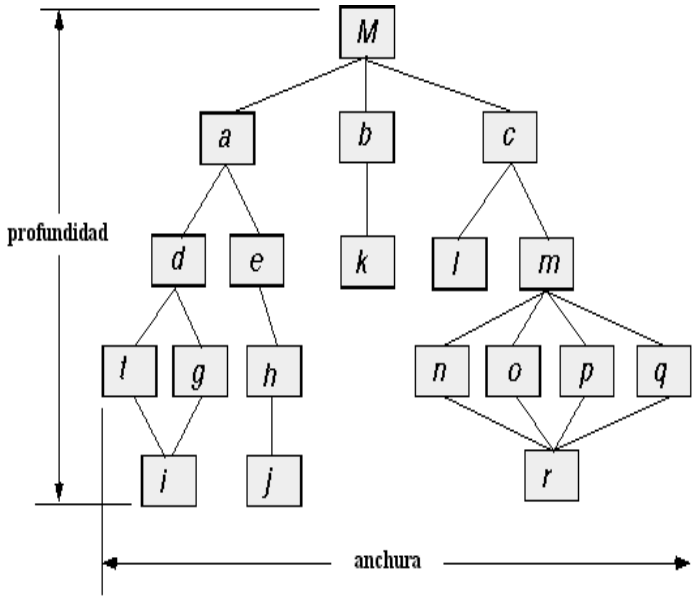
Proporciona al Arquitecto de Software estructuras de programas de fácil cambio y escalabilidad. Estilo más utilizado en sistemas de gran escala [LIB10].

Dentro de este estilo tenemos dos subestilos como son:

Programa principal/subprograma: Descompone sus funciones en una jerarquía de control donde el programa principal invoca a los otros programas subordinados, los cuales pueden a su vez invocar a otros.

18 **Ilustración tomada de:** Roger Pressman, Ingeniería del Software: Un enfoque práctico, McGraw Hill, 2001.

Llamada de procedimiento remoto: Los componentes de la arquitectura son distribuidos entre diferentes host de la red.



La ilustración número 8[19] muestra un ejemplo del estilo de arquitectura llamada y retorno.

1.8.4 ARQUITECTURAS ORIENTADAS A OBJETOS

Arquitectura en la cual los componentes del sistema encapsulan los datos y operaciones que son utilizadas para la manipulación de los mismos. La comunicación y coordinación entre componentes se realiza mediante el envío de mensajes [LIB10].

19 **Ilustración tomada de:** Roger Pressman, Ingeniería del Software: Un enfoque práctico, McGraw Hill, 2001.

En esta arquitectura podemos observar que:

- Los componentes del estilo se basan en: encapsulamiento, polimorfismo y herencia.
- Las interfaces están separadas de las implementaciones.
- Una interfaz puede ser implementada en varias clases.
- Los objetos interactúan a través de invocaciones de funciones y procedimientos.

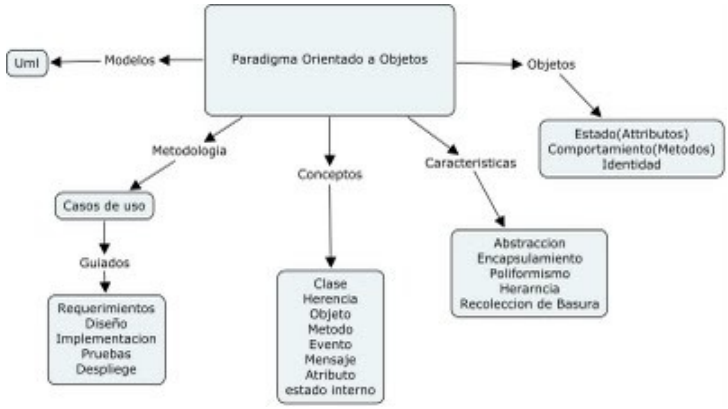


Ilustración 8: *Arquitectura Orientada a Objetos*

En la ilustración número 8[20] se muestra un ejemplo del estilo de arquitectura orientado a objetos.

1.8.5 ARQUITECTURAS ORIENTADAS A SERVICIOS

La Arquitectura Orientada a Servicios o SOA por sus siglas en inglés **S**ervice **O**riented **A**rchitecture establece un marco de diseño para la integración de aplicaciones independientes de tal manera que desde la red pueda accederse a sus funcionalidades, las cuales se ofrecen como servicios [WWW17].

20 **Ilustración tomada de:** http://elo-ge-ma.blogspot.com/2009_09_01_archive.html

La forma de implementarla es mediante *Servicios Web*, una tecnología basada en estándares e independiente de la plataforma, con la que SOA puede descomponer las aplicaciones monolíticas en un conjunto de servicios e implementar esta funcionalidad en forma modular.

Servicio: Es muy independiente de la plataforma donde se desarrolla y se lo orienta para la comunicación entre diferentes aplicaciones, generalmente de manera remota.

En la ilustración número 9[21] se muestra un ejemplo del estilo de arquitectura orientado a servicios.

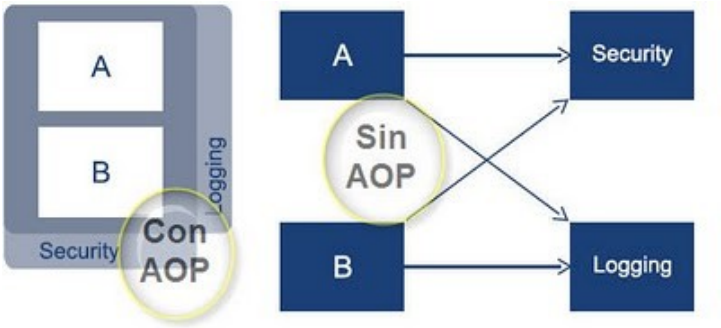
21 **Ilustración tomada de:** http://www.intellisoa.com/ar/index.php

1.8.6 ARQUITECTURAS ORIENTADAS A ASPECTOS

Esta arquitectura ayuda a mejorar el proceso de desarrollo de software, mediante la utilización del concepto de aspecto a lo largo de todo el ciclo de vida. Proporciona técnicas que permiten una identificación temprana de aspectos, su extracción, representación y posterior composición [WWW18].

En este paradigma, los aspectos se consideran como entidades de primera clase que pueden ser manipulados a lo largo del proceso de desarrollo de un sistema.

La definición de aspectos es un mecanismo de abstracción por el que éstos pueden incorporarse a un sistema existente de tal modo que los elementos que se añaden no quedan dispersos a lo largo de diversos módulos, sino que permanezcan en módulos independientes.

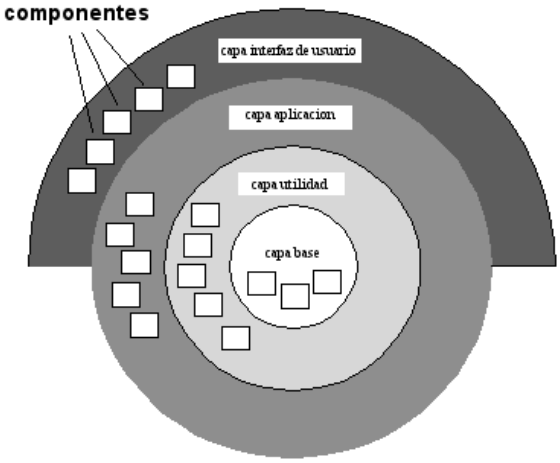


En la ilustración número 11[22] se muestra un ejemplo del estilo de arquitectura orientado a aspectos.

22 **Ilustración tomada de:** http://metodologiasdesistemas.blogspot.com/2008/06/aop-que-es-la-prog-orientada-aspectos.html

1.8.7 ARQUITECTURAS ESTRATIFICADAS

Esta arquitectura se encuentra estructurada en capas, en las cuales cada una realiza operaciones progresivas que cada vez más se aproximan al lenguaje de máquina [LIB10].



En la ilustración número 12[23] se muestra un ejemplo del estilo de arquitectura estratificada.

La estructura interna de este estilo es la siguiente:

- En la capa externa, los componentes sirven a las operaciones de interfaz de usuario.
- En la capa interna, los componentes realizan operaciones de interfaz del sistema.
- Las capas intermedias, proporcionan servicios de utilidad y funciones del software de aplicaciones.

23 **Ilustración tomada de:** Roger Pressman, Ingeniería del Software: Un enfoque práctico, McGraw Hill, 2001.

CAPÍTULO II

2 PATRONES DE ARQUITECTURA DE SOFTWARE

2.1 INTRODUCCIÓN

Cada patrón describe un problema concurrente en nuestro entorno, para describir después el camino a la solución a ese problema, de tal manera que pueda ser reutilizado en proyectos distintos [WWW19].

Esta idea de patrones, por su misma naturaleza genérica de Problema – Solución – Reutilización, se ha popularizado a nivel mundial, dentro de la ingeniería de software se ha extendido, tal así que se han creado y siguen creando catálogos de patrones para problemas en diferentes niveles de abstracción.

Generalmente, no es necesario inventar una nueva arquitectura de software para cada sistema de información. Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso en concreto. Entre los principales patrones arquitectónicos tenemos:

- Arquitectura Modelo Vista Controlador (Model – View – Controller).
- Arquitectura Tuberías y Filtros (Pipe and Filter).
- Arquitectura en Capas (Layered Systems).
- Arquitectura de Microkernel (Microkernel).

- Arquitectura Cliente Servidor y Sistemas de N – Capas (Client – Server and N – Tier Systems).
- Arquitectura de Repositorio (Repository).
- Arquitectura de Pizarra (Blackboard).
- Arquitectura de Maquina de Estado Finito (Finite State Machine).
- Arquitectura de Control de Procesos (Process Control).
- Arquitectura de Sistema Multiagente (Multi Agent System).
- Arquitectura Broker (Arquitectura Orientada a Servicios).
- Arquitectura Maestro Esclavo (Master Slave).
- Arquitectura de Intérprete (Interpreter – Virtual Machine).
- Arquitectura de Ejecutar y Hablar (Hub and Spoke).
- Arquitectura de Bus de Mensajes – Bus de Eventos (Message Bus – Event Bus).
- Arquitectura de Modelo Estructural (Structural Model).
- Arquitectura de Intercambio de Archivos (Peer to Peer).

De todos estos tomaremos para nuestro estudio los siguientes:

- Arquitectura en Capas.
- Arquitectura Tuberías y Filtros.
- Arquitectura de Pizarra.
- Arquitectura Broker.
- Arquitectura Modelo Vista Controlador.

2.2 DEFINICIONES

Es la solución a un problema de diseño especifico por medio de esquemas en contexto en particular. Proporciona un conjunto de subsistemas predefinidos, en los cuales, se definen reglas para la organización de las relaciones entre ellos.

Los patrones arquitectónicos son considerados como plantillas para arquitecturas de software especificas, que determinan las propiedades estructurales de una aplicación y tienen un impacto en la arquitectura de subsistemas.

"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo siquiera dos veces de la misma forma." [LIB11]

"Los patrones de diseño son la representación escrita de las técnicas que proporcionan la visión de alto nivel de un sistema de software. Un patrón es un conjunto de información que aporta la solución a un problema que se presenta en un contexto determinado. Para elaborarlo se aíslan sus aspectos esenciales y se añaden cuantos comentarios y ejemplos sean necesarios." [LIB12]

"Un patrón describe un problema de diseño recurrente, que surge en contextos específicos de diseño, y presenta un esquema genérico probado para la solución de este. El esquema de la solución describe un conjunto de componentes, responsabilidades y relaciones entre éstos, y formas en que dichos componentes colaboran entre sí." [LIB13]

2.3 ARQUITECTURA EN CAPAS

2.3.1 INTRODUCCIÓN

Este tipo de arquitectura es muy apropiada para aplicaciones de negocio con un amplio ciclo de vida o aplicaciones que tienen un comportamiento complejo. Mediante el énfasis del uso de interfaces para los contratos de comportamiento, se fuerza a la externalización de su infraestructura [WWW20].

La ilustración número 12[24] muestra la representación de la arquitectura tradicional en capas. Este es el tipo de arquitectura más básica y utilizada. Cada capa inferior depende de la capa superior, y a su vez cada capa va a depender de una infraestructura común y servicios comunes.

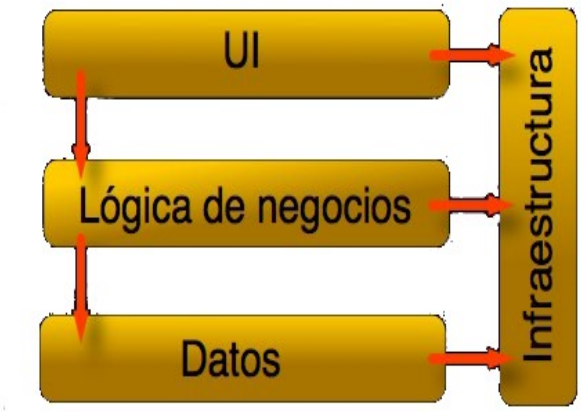


Ilustración 12: *Arquitectura en capas*

24 **Ilustración tomada de:** http://mario-chavez.blogspot.com/2008/07/la-arquitectura-de-cebolla-parte-1.html

2.3.2 DEFINICIÓN

"El patrón de arquitectura en Capas ayuda a estructurar aplicaciones que pueden descomponerse en grupos de subtareas en la que cada grupo de subtareas está en un nivel particular de abstracción [LIB13]*."*

2.3.3 ESTRUCTURA

Hay varias formas de implantar la filosofía de capas, dependiendo de cuál de los siguientes escenarios se espera cumplir:

1. Cada capa N+1 envía solicitudes de servicio a objetos de la capa inferior N, la cual a su vez se apoya en solicitudes a su capa inferior. Típicamente se produce una *cascada* de solicitudes, es decir para satisfacer una solicitud a una capa N+1, ésta requiere enviar solicitudes a la capa N [WWW21].

2. Cada capa N *notifica* a su capa superior N+1 de que a ocurrido algún evento de interés. La capa N+1 puede juntar varios eventos antes de notificar a su vez a su superior.

3. Como (1), se manejan solicitudes de forma *top – down (arriba hacia abajo)*, pero éstas no requieren llegar a la capa más interna o capa 1.

4. Como (2) pero las notificaciones no llegan hasta arriba.

5. Involucra dos pilas de N capas que se comunican entre sí. El ejemplo más conocido es el de los protocolos de en Redes de Computadoras .

2.3.4 VENTAJAS Y DESVENTAJAS

VENTAJAS:

- Reutilización de capas.
- Facilidad de estandarización.
- Dependencias se limitan a intra – capa.
- Contención de cambios a una o pocas capas.

DESVENTAJAS:

- A veces no se logra la contención del cambio y se requiere una cascada de cambios en varias capas.
- Pérdida de eficiencia.
- Trabajo innecesario por parte de capas más internas o redundante entre varias capas.
- Dificultad de diseñar correctamente la granularidad de las capas.

2.4 ARQUITECTURA TUBOS Y FILTROS

2.4.1 INTRODUCCIÓN

La popularidad de esta arquitectura se debe principalmente al sistema operativo Unix. Se ha convertido en popular por Ken Thompson[25] quien decidió limitar la arquitectura hacia una tubería lineal. El uso de dicha arquitectura fue idea de Doug McIlroy[26] su director en los laboratorios Bell en 1972 [WWW22].

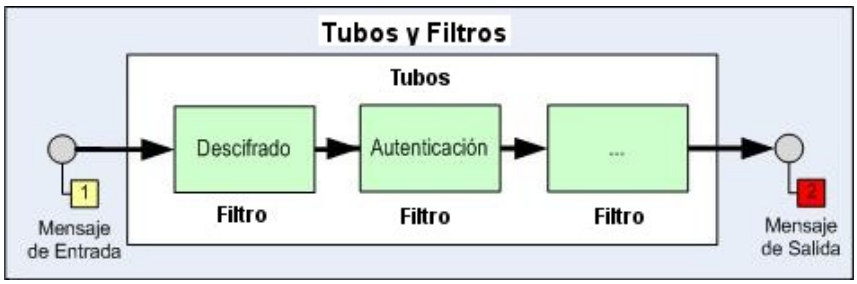


Ilustración 13: *Arquitectura Tubos y Filtros*

En la ilustración número 13[27] se muestra un ejemplo del patrón de arquitectura de tubos y filtros.

25 **Ken Thompson,** es un pionero en las ciencias de la computación. Su trabajo con el lenguaje de programación del lenguaje B y el sistema operativo UNIX y plan 9 para los laboratorios Bell. Se le adjudica a Thompso, junto a Dennis Ritchie, la creación de UNIX [WWW23].

26 **Doug McIlroy**, matemático, ingeniero y programador. El Dr. McIlroy ha pasado a la historia como creador de: la segmentación por medio de filtros y 'pipelines', básica en la implementación de UNIX, el actual concepto de componentes en software y; varios comandos y herramientas de UNIX como spell, diff, sort, join, graph, speak, tr, etc [WWW24].

27 **Ilustración tomada de:** http://blogs.technet.com/pablojunco/

2.4.2 DEFINICIÓN

"El patrón de arquitectura Tuberías y Filtros proporciona una estructura para los sistemas de flujo de proceso de datos. Cada paso del proceso se encapsula en un componente de filtro. Los datos se pasan a través de tubos entre filtros adyacentes. Recombinar filtros le permite construir familias de sistemas relacionados [LIB13]*"*.

2.4.3 ESTRUCTURA

El patrón de arquitectura Tuberías y Filtros divide la tarea de un sistema en varios pasos de procesamiento secuencial. Estos pasos están conectados por un flujo de datos a través del sistema, los datos de salida de un paso es la entrada de la fase siguiente.

Cada paso del proceso es ejecutado por un componente de filtro. Un filtro consume y entrega datos de forma incremental en contraste con el consumo de todas sus aportaciones antes de producir cualquier salida para lograr una baja latencia y permitir el procesamiento paralelo real.

La entrada al sistema es proporcionada por una fuente de datos como un archivo de texto. La salida de flujos en un sumidero de datos como un archivo, terminal, programa de animación, etc.

La fuente de datos, los filtros y el sumidero se conectan en forma secuencial por tuberías. Cada tubo implementa el flujo de datos entre los pasos de procesamiento adyacentes. La secuencia de filtros combinados por tuberías se denomina línea de proceso [WWW25].

El patrón de arquitectura Tubos y Filtros esta conformado por:

- Tubo.
- Filtro.
- Bomba (Entrada).
- Sumidero (Salida).

2.4.3.1 TUBO

Es el conector que pasa los datos de un filtro al siguiente. Se trata de un flujo unidireccional de datos, que suele ejecutarse por un buffer de datos para almacenar todos los datos; hasta que el siguiente filtro tiene tiempo para procesarlo [WWW26].

2.4.3.2 FILTRO

Es el encargado de filtrar o transformar los datos que recibe a través de las tuberías [WWW26].

2.4.3.3 BOMBA

También llamado productor es el origen de datos. Puede ser archivos, bases de datos, o dispositivos de entrada, creando continuamente nuevos datos [WWW26].

2.4.3.4 SUMIDERO

Se conoce también como consumidor es el objetivo de los datos, puede ser a archivos, bases de datos o dispositivos de salida [WWW26].

2.4.4 VENTAJAS Y DESVENTAJAS

VENTAJAS:

- Permite entender el sistema global en términos de la combinación de componentes.
- Soporta de una buena manera la reutilización.
- Facilidad de mantenimiento y mejora.
- Facilidad de diagnóstico.
- Soportan la ejecución concurrente.

DESVENTAJAS:

- No aconsejable para cuando se necesita interactividad.
- Problemas de rendimiento ya que los datos se transmiten en forma completa entre filtros.

2.5 ARQUITECTURA EN PIZARRA

2.5.1 INTRODUCCIÓN

Imagine un número de estudiantes que están escribiendo en la pizarra al mismo tiempo tratando de resolver un problema. ¿Crees que lo hará?. Y si le añadimos un maestro que primero le pide a cada estudiante lo que quiere escribir en la pizarra y decide qué estudiante tiene la mejor idea.

Y cuando el estudiante haya terminado, el proceso se repite. Esta es la idea detrás de la arquitectura en pizarra. Los estudiantes se llaman fuentes de conocimiento. El profesor se denomina *"intérprete de comandos de control"* [WWW27].

2.5.2 DEFINICIÓN

"El patrón de arquitectura en Pizarra es útil para problemas para los que no se conocen las estrategias de solución determinista. En la Arquitectura en Pizarra varios subsistemas especializados pueden reunir sus conocimientos para construir una posible solución parcial o aproximada [LIB13]*."*

2.5.3 ESTRUCTURA

La idea detrás de la Arquitectura de Pizarra es una colección de programas independientes que trabajan conjuntamente en una estructura de datos común. Cada programa especializado trabaja para la solución de una parte específica de la tarea general, y todos los programas trabajan juntos en la solución total.

Estos programados especializados son independientes uno de otro. Ellos no los llaman unos a otros, ni hay una secuencia predeterminada para su activación. En cambio, la dirección tomada por el sistema es determinado principalmente por el estado actual de progreso.

Un componente central de control evalúa el estado actual de la transformación y coordinación de los programas especializados. Estos datos dirigidos del régimen de control hace posible la experimentación con diferentes algoritmos, y permite experimentar los derivados de la heurística para el control de su procesamiento como se muestra en la ilustración número 14[28] [WWW28].

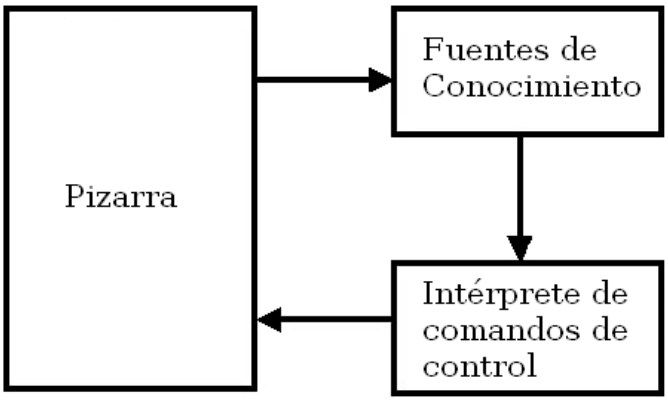


Ilustración 14: *Arquitectura de Pizarra*

Un sistema de pizarra tiene frecuentemente la siguiente estructura:

- Fuente de conocimiento.
- Pizarra.
- Intérprete de comandos de control.

28 **Ilustración tomada de:** http://www.dossier-andreas.net/software_architecture/

2.5.3.1 FUENTE DE CONOCIMIENTO

La fuente de conocimiento es un componente que se añade a la solución del problema. Puede ser cualquier cosa que se lee de un cierto nivel de pizarra, y propone una modificación a los componentes de la pizarra. Su forma más común es una regla de producción. Una fuente de conocimiento es totalmente ajena a otras fuentes de conocimiento [WWW28]

2.5.3.2 PIZARRA

La pizarra es la estructura de datos común de las fuentes de conocimiento. La pizarra es capaz de representar a todos los estados de algún espacio del problema. La pizarra contiene varios niveles de descripción con respecto al espacio del problema. Estos niveles pueden tener varias relaciones con los demás. Los niveles son partes de la misma estructura de datos. Si se necesita estructura de datos por separado, la pizarra se divide en paneles. Cada panel a su vez puede contener varios niveles [WWW28].

2.5.3.3 INTÉRPRETE DE COMANDOS DE CONTROL

El intérprete de comandos de control determina que fuente de conocimiento tiene la oportunidad de cambiar la pizarra. Cada ciclo de ejecución, identifica los cambios a la pizarra, activa las fuentes de conocimiento apropiadas, selecciona una de estas y la ejecuta [WWW28].

La ilustración número 15[29] muestra un modelo de arquitectura de pizarra dividida en paneles.

29 **Ilustración tomada de:** http://www.dossier-andreas.net/software_architecture/

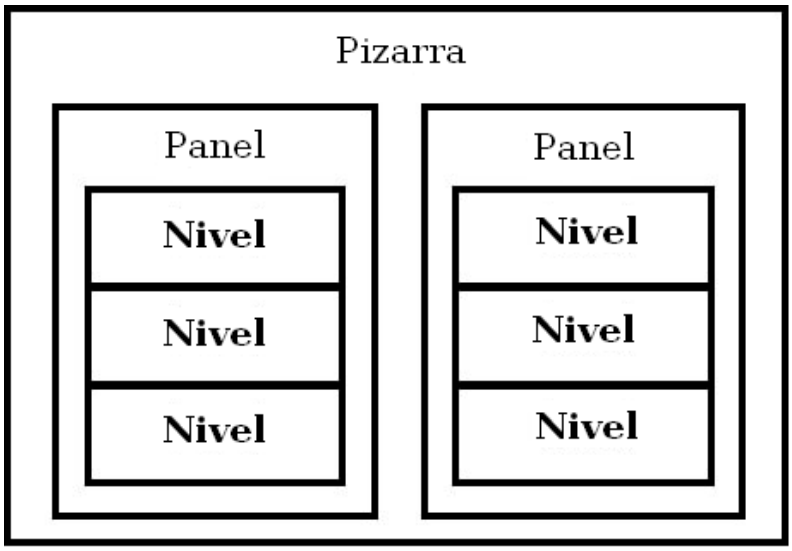


Ilustración 15: *Arquitectura de Pizarra dividida en Paneles*

2.5.4 VENTAJAS Y DESVENTAJAS

VENTAJAS:

- Flexibilidad de la configuración.
- Flexibilidad en la solución de problemas.
- Selección de las fuentes de conocimiento.
- Solucionador de problemas múltiples.

DESVENTAJAS:

- No hay lenguaje de comunicación.
- La complejidad computacional de la cooperación.
- Dificultad en la adquisición del conocimiento.

2.6 ARQUITECTURA DE BROKER

2.6.1 INTRODUCCIÓN

Es un patrón arquitectónico aplicado a la estructuración de sistemas distribuidos, en los cuales es necesaria la interacción remota de componentes altamente desacoplados. Lo anterior se logra al introducir un componente Broker cuya función principal es lograr el desacoplamiento de los clientes y de los servidores [WWW29].

También registra a los servidores, logrando de esta forma que los servicios que estos ofrecen estén disponibles a los posibles clientes.

2.6.2 DEFINICIÓN

"El patrón de arquitectura Broker puede ser usado para estructurar sistemas de software distribuidos con los componentes que interactúan disociada por las llamadas de servicio remoto. Un componente Broker es responsable de la coordinación de las comunicaciones, tales como solicitudes de reenvío, así como para la transmisión de los resultados y excepciones [LIB13]*."*

Al patrón de arquitectura Broker también se lo conoce como *"Arquitectura Orientada a Servicios"* y es utilizado por CORBA (Common Object Request Broker Architecture), Servicios Web (Web Services) y Jini (Sistemas Distribuidos en Java).

2.6.3 ESTRUCTURA

Los servidores se registran en el broker, y ponen sus servicios a disposición de los clientes a través de interfaces de método. Los clientes tienen acceso a la funcionalidad de los servidores mediante el envío de peticiones a través del Broker como se muestra en la ilustración número 16[30] [WWW30].

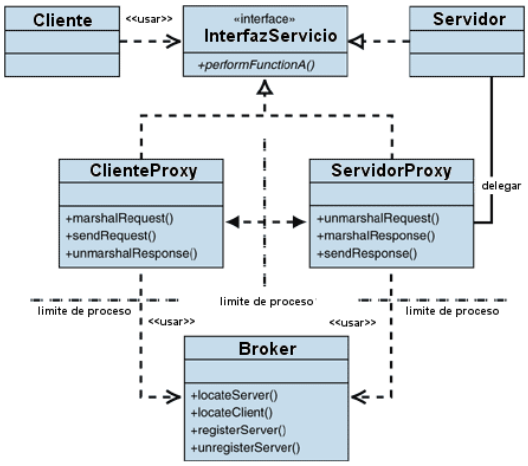


Ilustración 16: *Arquitectura Broker*

El patrón Broker comprende de seis componentes participantes:

- Cliente.
- Servidor.
- Broker.
- Puentes.
- Proxy del lado del cliente.
- Proxy del lado del servidor.

30 **Ilustración tomada de:** http://msdn.microsoft.com/en-us/library/ms978706.aspx

2.6.3.1 CLIENTE

Los clientes son aplicaciones que acceden a los servicios de los servidores. Para invocar servicios remotos, los clientes envían solicitudes al *broker*. Después que la operación se ha ejecutado, los clientes reciben respuestas o excepciones del *broker*.

La interacción entre clientes y servidores se basa en un modelo dinámico, lo cual significa que los servidores también pueden actuar como clientes. Los clientes no necesitan conocer la ubicación de los servidores que acceden, permitiendo de esta manera añadir nuevos servicios y que los ya existentes puedan moverse a otras ubicaciones [WWW31].

2.6.3.2 SERVIDOR

Un servidor implementa objetos que exponen su funcionalidad a través de interfaces que consisten de operaciones y atributos. Las interfaces estarán disponibles a través de un lenguaje de definición de interfaz (IDL) o un estándar binario [WWW31].

Hay dos tipos de servidores:

1. Servidores que ofrecen servicios comunes a muchos dominios de aplicación.

2. Servidores que implementan una funcionalidad especifica para un dominio de aplicación particular.

2.6.3.3 BROKER

Un *broker* es un mensajero, responsable de la transmisión de solicitudes de clientes a servidores, así como la transmisión de respuestas y excepciones de servidores a clientes.

Localiza al receptor de una solicitud basándose en un sistema de identificadores únicos. Ofrece API's a clientes y servidores que incluyen operaciones para el registro de servidores, y la invocación de métodos de servidores.

Cuando llega una solicitud dirigida a un servidor que es mantenido por el broker local, éste pasa la solicitud directamente al servidor. Si el servidor esta inactivo, el broker lo activa. El broker envía todas las respuestas y excepciones, producto de la ejecución de un servicio, al cliente que hizo la solicitud. Si el servidor especificado es mantenido por otro broker, el broker local encuentra una ruta al broker remoto y le envía la solicitud [WWW31].

2.6.3.4 PUENTE

Los puentes son componentes opcionales utilizados para esconder los detalles de implementación cuando dos brokers interoperan. Supóngase que un sistema Broker se ejecuta en una red heterogénea. Si se transmiten solicitudes sobre la red, se deben comunicar brokers diferentes independientemente de las redes y de los sistemas operativos utilizados [WWW31].

2.6.3.5 PROXY DEL LADO DEL CLIENTE

Los proxies del lado del cliente representan una capa adicional entre los clientes y el broker, para proveer transparencia en el sentido que un objeto remoto aparece como local ante el cliente, es decir, oculta los detalles de implementación tales como [WWW31]:

- El mecanismo de comunicación entre procesos, utilizado para transferir un mensaje entre clientes y brokers.
- La creación y eliminación de bloques de memoria.
- El marshaling[31] de parámetros y resultados.

2.6.3.6 PROXY DEL LADO DEL SERVIDOR

Los proxies del lado del servidor generalmente son análogos a los proxies de lado del cliente, la diferencia es que son responsables de recibir solicitudes, desempaquetar los mensajes de entrada, el unmarshaling de los parámetros, llamar al servicio apropiado, y el marshaling de resultados y excepciones antes de enviarlos al cliente [WWW31].

2.6.4 VENTAJAS Y DESVENTAJAS

VENTAJAS:

- Transparencia de ubicación.
- Facilidad para modificar un componente sin afectar al resto.

31 Marshaling, Lenguaje IDL Implementaciones cliente-servidor [WWW32]

- Portabilidad.
- Interoperatibilidad entre distintos tipos de brokers.
- Reusabilidad.

DESVENTAJAS:

- Eficiencia restringida.
- Menor tolerancia frente a fallos: si falla el broker, todo el sistema se viene abajo.
- Prueba y depuración costosas, pues existen muchos componentes en el sistema.

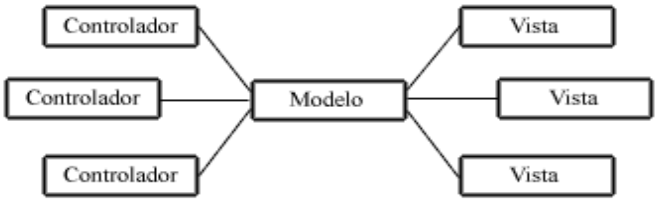
2.7 ARQUITECTURA MODELO VISTA CONTROLADOR

2.7.1 INTRODUCCIÓN

La arquitectura MVC (Modelo Vista Controlador) fue introducida como parte de la versión Smalltalk-80[32] del lenguaje de programación Smalltalk en 1979 por Tryge Reenskaug en los laboratorios de investigación de Xerox. Fue diseñada para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos.

Sus características principales son que el Modelo, las Vistas y los Controladores se tratan como entidades separadas; esto se hace que cualquier cambio producido en el Modelo se refleje automáticamente en cada una de las Vistas [WWW33].

En la ilustración número 17[33], vemos la arquitectura MVC en su forma más general. Hay un Modelo, múltiples Controladores que manipulan ese Modelo, y hay varias Vistas de los datos del Modelo, que cambian cuando cambia el estado de ese Modelo.



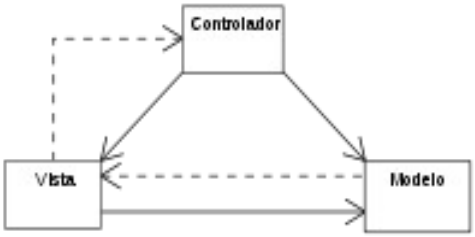
32 La implementación original esta descrita a fondo en *Programación de Aplicaciones en Smalltalk-80(TM): Como utilizar Modelo Vista Controlador.*

33 **Ilustración tomada de:** http://sunsite.dcc.uchile.cl/java/docs/JavaTut/Apendice/

2.7.2 DEFINICIÓN

"El patrón de arquitectura Modelo Vista Controlador (MVC) divide una aplicación interactiva en tres componentes. El modelo contiene la funcionalidad básica y datos. La vista muestra la información al usuario. El controlador maneja la entrada del usuario. La vista y el controlador en conjunto constituyen la interfaz de usuario. Un cambio en el mecanismo de propagación garantiza la coherencia entre la interfaz de usuario y el modelo [LIB13]*."*

A continuación se muestra en la ilustración número 18[34] la relación existente entre el modelo, la vista y el controlador en donde las líneas solidas indican una asociación directa, y las punteadas una indirecta.



La idea fundamental de MVC es construir una aplicación separando de forma clara las capas de Modelo (Datos), Vista (Presentación) y Controlador (Funciones, métodos …) para reducir el acoplamiento entre la lógica de negocio y la de presentación.

34 **Ilustración tomada de:** http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

2.7.3 ESTRUCTURA

La estructura básica de este patrón de arquitectura es el siguiente:

- Modelo.
- Vista.
- Controlador.

2.7.3.1 MODELO

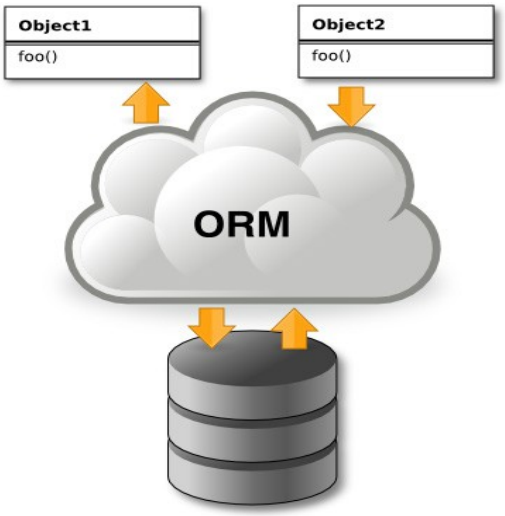
Es el objeto que representa los datos del programa[35]. Maneja los datos y controla todas sus transformaciones. El modelo no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos.

Es el propio sistema el que tienen encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo [WWW33].

2.7.3.1.1 ORM (MAPEO OBJETO – RELACIONAL):

Es una técnica de programación que nos permite realizar la conversión de tipos de datos entre los lenguajes de programación orientados a objetos y los utilizados en las bases de datos relacionales, es decir, las tablas de nuestra base de datos pasan a ser clases y los registros objetos que podemos manejar con mayor facilidad [WWW34].

35 La mayoría de sistemas utilizan un Sistema de Gestión de Base de Datos para gestionar los datos en esta capa.



En la ilustración número 19[36] se muestra un diagrama representativo del mapeo objeto relacional.

TIPO DE DATO (BDD)	TIPO DE DATO (POO)
CHAR, VARCHAR, VARCHAR2	String
NUMBER, INTEGER, INT	Integer
MONEY, DOUBLE, NUMERIC, REAL	double
FLOAT	float
DATE, DATETIME, TIME	Date
BOOLEAN	boolean

Tabla 4: *Tabla de conversión de tipos de datos entre base de datos relacionales y lenguajes de programación orientados a objetos*

36 **Ilustración tomada de:** http://www.neurosoftware.ro/programming-blog/wp-content/plugins/wp-o-matic/cache/73bac_orm.jpg

Entre las ventajas más comunes de la utilización del Mapeo Objeto Relacional tenemos:

Reutilización: *L*a principal ventaja que aporta un ORM es la reutilización permitiendo llamar a los métodos de un objeto de datos desde distintas partes de la aplicación e incluso desde las diferentes aplicaciones.

Encapsulación: La capa ORM encapsula la lógica de los datos pudiendo hacer cambios que afectan a toda la aplicación únicamente modificando una función.

Portabilidad: Utilizar una capa de abstracción nos permite cambiar en mitad de un proyecto de una base de datos MySQL a una Oracle sin ningún tipo de complicación. Esto es debido a que no utilizamos las sintaxis de las bases de datos para acceder a nuestro modelo, sino una sintaxis propia del ORM que es capaz de traducir a diferentes tipos de base de datos.

Seguridad: Los ORM implementan mecanismos de seguridad que protegen a nuestra aplicación de los ataques mas comunes como son los SQL injections.

Mantenimiento del código: Gracias a la correcta ordenación de la capa de datos, modificar y mantener nuestro código es una tarea sencilla [WWW35].

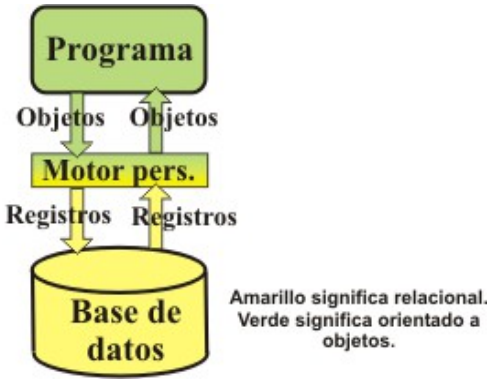
2.7.3.1.2 PERSISTENCIA

Acción de preservar la información de un objeto de forma permanente (guardar), pero a su vez también se refiere a poder recuperar la información del mismo (leer) para que pueda ser nuevamente utilizada.

La persistencia permite al programador almacenar, transferir y recuperar el estado de los objetos. Para esto tenemos las siguiente técnicas:

Serialización: Es el proceso de codificación de un objeto en un medio de almacenamiento (archivo o *buffer* de memoria).

Motor de persistencia: Traduce entre los dos formatos de datos de registros a objetos y de objetos a registros.



La situación se ejemplifica en la ilustración número 21[37], cuando el programa quiere grabar un objeto llama al motor de persistencia que traduce el objeto a registros y llama a la base de datos para que guarde estos registros.

De la misma manera, cuando el programa quiere recuperar un objeto, la base de datos recupera los registros correspondientes, los cuales son traducidos en formato de objeto por el motor de persistencia [WWW36].

37 **Ilustración tomada de:** http://www.ufg.edu.sv/ufg/theorethikos/Julio04/mdp6.html

Ejemplos de motores de persistencia para distintos lenguajes:

Java:

- Hibernate.
- Ibatis.
- Java Data Objects (JDO).
- OpenJPA.
- TopLink.

.NET:

- DataObjects.NET.
- iBatis.NET.
- Nhibernate.

PHP:

- Doctrine.
- Propel.

2.7.3.2 VISTA

Es el objeto que maneja la presentación visual (Interfaz Gráfica de Usuario[38]) de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa con el Modelo a través de una referencia al propio Modelo.

38 **GUI (Interfaz Gráfica de Usuario),** wikipedia define la interfaz gráfica de usuario como "un método de interacción con un ordenador a través del paradigma de manipulación directa de imágenes gráficas, controles y texto".

Una vista obtiene del modelo solamente la información que necesita para desplegar y se actualiza cada vez que el modelo del dominio cambia por medio de notificaciones generadas por el modelo de la aplicación [WWW33].

Las vistas son responsables de:

- Recibir datos del modelo y los muestra al usuario.

- Tener un registro de su controlador asociado (normalmente porque además lo instancia).

- Pueden dar el servicio de *"Actualización()"*, para que sea invocado por el controlador o por el modelo (cuando es un modelo activo que informa de los cambios en los datos producidos por otros agentes) [WWW37].

La capa de la vista también puede aprovechar la separación de código. Las páginas web suelen contener elementos que se muestran de forma idéntica a lo largo de toda la aplicación: cabeceras de las páginas, pie de página y su navegación global. Normalmente solo se cambia el interior de la página.

Muchas interfaces gráficas de usuario, como Swing[39] o MFC[40], hacen innecesario el uso de un controlador ya que definen su propio flujo de control y manejan los eventos internamente; integran así la vista y el controlador. A esta variante se la suele denominar Document – View.

39 **Swing,** es una plataforma independiente, MVC – GUI framework para java. Sigue un simple modelo de programación por hilos [WWW38].

40 **MFC (Microsoft Foundation Class),** es un conjunto de clases que proveen un acceso más sencillo a las API's de Windows [WWW39].

2.7.3.3 CONTROLADOR

Es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. Interactúa con el Modelo a través de una referencia al propio Modelo [WWW33].

La unión entre la capa de presentación y capa de negocio conocido como el paradigma de la Programación por capas representaría la integración entre Vista y su correspondiente Controlador de eventos y acceso a datos. El flujo que sigue el control es el siguiente como se muestra la ilustración 21[41]:

1. El usuario interactúa con la interfaz de usuario de alguna forma (el usuario pulsa un botón, enlace, etc.).

2. El controlador recibe (por parte de la interfaz – vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler[42]) o callback[43].

41 **Ilustración tomada de:** http://www.tufuncion.com/mvc

42 **Handler**, manipulador, rutina de software que realiza una determinada tarea [WWW41].

43 **Callback**, es una porción de código ejecutable que es pasado como argumento a otro código [WWW42].

3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carrito de compra del usuario). Los controladores complejos están a menudo estructurados en un patrón de comando que encapsula las acciones y simplifica su extensión.

4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se reflejan los cambios en el modelo (por ejemplo, produce un listado del contenido del carrito de compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador puede ser utilizado para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aún así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice.

   Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente [WWW40].

2.7.4 FRAMEWORKS

La traducción literal al español sería “Marco de trabajo”, básicamente es una estructura de desarrollo sobre la cual podemos programar, es un software que ayuda a desarrollar las cosas de una manera más rápida sin empezar desde cero [WWW43].

Los frameworks ayudan en el desarrollo de software ya que tienen una estructura definida para la creación de aplicaciones con mayor rapidez. Se utiliza programación orientada a objetos, permitiendo la reutilización de código.

Ilustración 22: *Frameworks MVC*

En la ilustración número 22[44] se muestra los logos de un grupo de los varios frameworks que utilizan la arquitectura MVC para su desarrollo. A continuación se presenta una descripción de los diferentes frameworks que utilizan la arquitectura MVC para su funcionamiento con su respectiva licencia, organizados por el lenguaje de programación que utilizan:

FRAMEWORKS PARA RUBY:

44 **Ilustración tomada de:** http://ateneatech.com/blog/category/frameworks

LENGUAJE	LICENCIA	NOMBRE
Ruby	MIT[45]	Ruby on Rails

Tabla 5: *Frameworks MVC para Ruby*

FRAMEWORKS PARA JAVA:

LENGUAJE	LICENCIA	NOMBRE
Java	Apache[46]	Struts
Java	Apache	Beehive
Java	Apache	Spring
Java	Apache	Tapestry
Java	Apache	Aurora
Java	Apache	Java Server Faces
Java	Apache	Jboss Seam

Tabla 6: *Frameworks MVC Java*

FRAMEWORKS MVC PARA PERL:

LENGUAJE	LICENCIA	NOMBRE
Perl	GPL	Catalyst
Perl	GPL	CGI: Application
Perl	GPL	Gantry Framework
Perl	GPL	Jifty

45 **MIT (Massachusetts Institute of Technology),** esta licencia permite reutilizar el software así licenciado tanto para ser software libre como para ser software privativo, permitiendo no liberar los cambios realizados al programa original [WWW44].

46 **Apache (Apache License o Apache Software License para versiones anteriores a 2.0),** es una licencia de software libre creada por la Apache Software Foundation (ASF). La licencia Apache (con versiones 1.0, 1.1 y 2.0) requiere la conservación del aviso de copyright y disclaimer, pero no es una licencia copyleft, ya que permite el uso y la distribución del código fuente para software libre y software privativo [WWW45].

Perl	GPL[47]	Maypole
Perl	GPL	OpenInteract 2
Perl	Comercial	PageKit
Perl	GPL	Cyclone 3
Perl	ECL[48]	Solscite
Perl	GPL	CGI: Builder

Tabla 7: *Frameworks MVC para Perl*

FRAMEWORKS MVC PARA PHP:

LENGUAJE	LICENCIA	NOMBRE
PHP	LGPL[49]	Agavi
PHP	BSD[50]	Zend Framework
PHP	MIT	CakePHP
PHP	GPL	Codelgniter
PHP	Otra	Kohana

47 **GPL (Licencia General Pública de GNU),** llamada comúnmente GNU GPL, la usan la mayoría de los programas de GNU y más de la mitad de las aplicaciones de software libre [WWW46].

48 **ECL (Educational Community License),** permite trabajar sobre el software pero informando después de sus modificaciones y novedades [WWW47].

49 **LGPL (Licencia General Pública Reducida de GNU),** la usan algunas, pero no todas, las bibliotecas GNU. Esta licencia fue llamada en un principio *GPL para bibliotecas* pero le cambiamos el nombre debido a que animaba a la gente a utilizar esta licencia más de lo debido [WWW46].

50 **BSD (Berkley Software Distribution),** pertenece al grupo de licencias de software libre. Esta licencia tiene menos restricciones en comparación de otras como la GPL estando muy cercana al dominio público. La licencia BSD al contrario que la GPL permite el uso de código fuente en software no libre [WWW46].

PHP	MPL 1.1[51]	PHP 4 ECORE
PHP	MIT	Symfony

Tabla 8: *Frameworks MVC para PHP*

FRAMEWORKS MVC PARA PYTHON:

LENGUAJE	LICENCIA	NOMBRE
Python	Varias	Turbogears
Python	BSD	Django

Tabla 9: *Frameworks MVC para Python*

FRAMEWORKS MVC PARA .NET:

LENGUAJE	LICENCIA	NOMBRE
.NET	Castle Project	Monorail
.NET	Apache	Spring .NET
.NET	Apache	Maverick .NET
.NET	Microsft Patterns & Practices	User Interface Process (UIP) Application Block

Tabla 10: *Frameworks MVC para .NET*

2.7.5 VENTAJAS Y DESVENTAJAS

VENTAJAS:

- Existe una clara separación de todos los componentes que se utilizan en el desarrollo.

51 **MPL (Licencia Pública de Mozilla),** es una licencia de código abierto y software libre. Fue desarrollada originalmente por Nestcape Communications Corporation – una división de la compañía America Online – y mas tarde su control fue traspasado a la Fundación Mozilla [WWW48].

- Crea independencia en el funcionamiento.
- Facilita el mantenimiento de la aplicación.

DESVENTAJAS:

- La separación en capas agrega complejidad al desarrollo.
- La cantidad de archivos que se necesitan mantener y depurar pueden ser muchos.
- Es una tecnología en alto crecimiento por lo que su aprendizaje puede ser más complejo.

2.8 RESUMEN DE LOS PATRONES DE ARQUITECTURA DE SOFTWARE

A continuación se detallan los principales patrones arquitectónicos:

PATRONES	DEFINICIÓN	ATRIBUTOS ASOCIADOS	ATRIBUTOS EN CONFLICTO	UTILIZACIÓN
Capas	Organización de un sistema que es descompuesto en subtareas de acuerdo a un único nivel de abstracción.	Reusabilidad Portabilidad Facilidad de Prueba	Desempeño Mantenibilidad	Sistemas Operativos Protocolos de Comunicación
Tubos y Filtros	Los filtros ayudan a encapsular cada paso de procesamiento, mientras que los tubos dan paso a los datos entre filtros adyacentes.	Reusabilidad Mantenibilidad	Desempeño	Sistemas Operativos
Pizarra	Usado para encontrar una solución con estrategias no determinísticas, es decir una solución parcial o aproximada.	Modificabilidad Mantenibilidad Reusabilidad Integridad	Desempeño Facilidad de Prueba	Inteligencia Artificial
Broker	Utilizado en sistemas de software distribuido con componentes desencajados que interactúan por invocaciones a sistemas remotos, es responsable de coordinar la comunicación, como el reenvío de solicitudes, así como también la transmisión de resultados y excepciones.	Modificabilidad Portabilidad Reusabilidad Escabilidad Interoperatibilidad	Desempeño	Sistemas Distribuidos

Modelo Vista Controlador	Posee tres partes: Modelo, esta compuesto de información principal y además de datos. Vistas, poseen información para el usuario. Controladores, capturan la entrada del usuario.	Funcionabilidad Mantenibilidad	Desempeño Portabilidad	Aplicaciones con sofisticadas interfaces Aplicaciones grandes y complejas

Tabla 11: *Descripción de los principales patrones arquitectónicos*[52]

52 Información basada en el libro de PRESSMAN R. (2002) Ingeniería de Software. Un Enfoque Práctico. Quinta Edición; en la cual se detallan los principales estilos arquitectónicos que se pueden encontrar en software construido para sistemas basados en computadoras.

CAPÍTULO III

3 ARQUITECTURAS TECNOLÓGICAS

3.1 INTRODUCCIÓN

El propósito es definir las mejores clases tecnológicas necesarias para proveer un ambiente para las aplicaciones que manejan los datos. Estas clases de tecnología referidas como plataformas apoyarán al negocio con un ambiente de datos compartidos.

Para el manejo de los datos del negocio las plataformas tecnológicas proveen los medios para colectar los datos desde los proveedores de datos, transportarlos, almacenarlos y procesarlos. Además de entregarlos a los consumidores de ellos [WWW49].

En este capítulo realizaremos el estudio de dos las arquitecturas tecnológicas más utilizadas como son:

- JEE (Java Enterprise Edition).
- .NET (Microsoft).
- PHP

3.2 JEE (JAVA ENTERPRISE EDITION)

3.2.1 INTRODUCCIÓN

Java Enterprise Editon anteriormente conocida como J2EE (Java Second Enterprise Edition) fue desarrollada por Sun Microsystems[53], iniciando con la liberación de J2EE 1.3, la especificación fue desarrollada bajo el Java Community Process[54]. JSR 58 específica J2EE 1.3 y JSR 151 específica J2EE 1.4.

El SDK[55] de J2EE 1.3 fue liberado como beta en abril 2001 y J2EE 1.4 en diciembre de 2002, posteriormente Java EE 5 fue desarrollada bajo el JSR 244, su liberación se produce el 11 de mayo de 2006; y finalmente Java EE 6 que fue desarrollada bajo el JSR 316 con su liberación el 10 de diciembre de 2009 [WWW50].

53 **Sun Microsystems**, es una empresa informática de Silicon Valley, fabricante de semiconductores y software; hoy de propiedad de la Oracle Corporation tras una compra por 7.400 millones de dólares [WWW51].

54 **Java Community Process**, el proceso de la comunidad Java, o Java Community Process JCP, establecido en 1998, es un proceso formalizado el cual permite a las partes interesadas a involucrarse en la definición de las futuras versiones y características de la plataforma Java. El proceso JCP conlleva el uso de Java Specification Request JSR (Especificación de Solicitudes de Java), las cuales son documentos formales que describen las especificaciones y tecnologías propuestas para que sean añadidas a la plataforma Java [WWW52].

55 **Software Development Kit (SDK)**, o Kit de Desarrollo de Software es generalmente un conjunto de herramientas de desarrollo que le permite a un programador crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, frameworks, plataformas de hardware, ordenadores, videoconsolas, etc [WWW53].

3.2.2 DEFINICIÓN

Según el sitio web oficial de Sun Microsystems: *"Java EE es una arquitectura tecnológica que se basa en la sólido fundamento de Java Plataform, Standard Edition (J2SE) y es el estándar de la industria para el desarrollo de aplicaciones empresariales, aplicaciones orientadas a servicios (SOA) y aplicaciones web de próxima generación."*

Según el sitio web oficial de Sun Microsystems la definición de JEE versión 6 es: *"Java Plataform, Enterprise Edition (Java EE) 6 es el estándar de la industria para la informática empresarial de Java. Utilizar el nuevo y ligero Java EE 6 Web Profile para crear aplicaciones web de próxima generación, y todo el poder de Java EE 6 Plataform para aplicaciones empresariales. Los desarrolladores se beneficiarán de las mejores de la productividad con más anotaciones, más POJO's[56], la simplificación de paquete, y menos configuración de XML.* [WWW54]*"*

Las mejoras que incluye JEE 6 son: la inclusión de perfiles, los cuales nos permite utilizar solo ciertos fragmentos de toda la especificación (Perfil Completo y Perfil Web). Actualización de JAX-RS y JAX-WS[57], JPA (Java Persistence Api)[58] 2.0 con nuevas librerías, Servlets 3.0 los cuales incluyen varios POJO's, EJB (Enterprise Java Beans) 3.1; entre los más importantes.

56 **Plain Old Java Object (POJO)**, es una sigla creada en septiembre de 2000 y utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un framework en especial.

57 **JAX-RS y JAX-WS**, son los estándares utilizados para el manejo de los servicios web de java.

58 **JPA (Java Persistence Api)**, es el API de Java para el manejo de la persistencia.

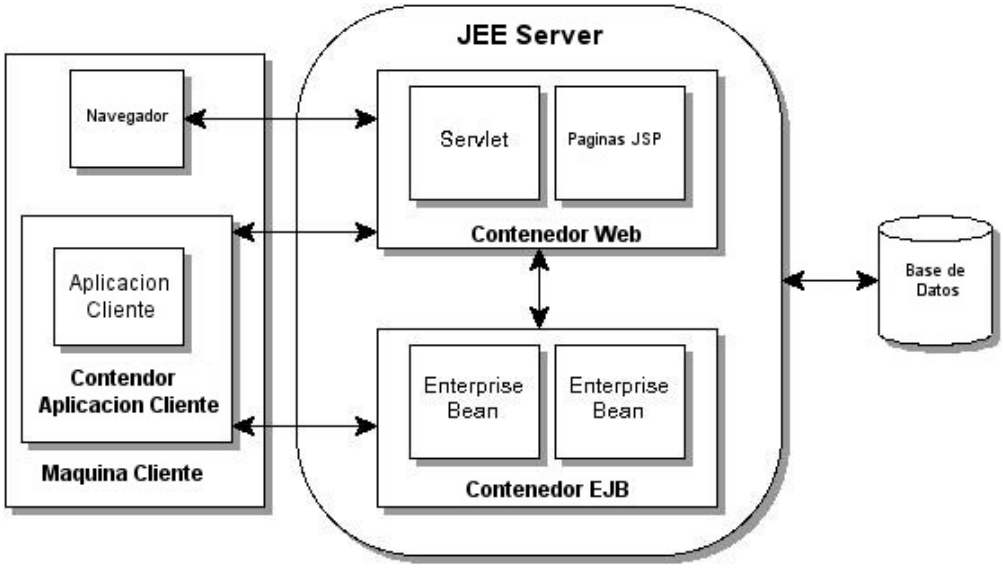


Ilustración 23: *Arquitectura Tecnológica JEE*

La ilustración número 23[59] muestra la disposición de la arquitectura tecnológica Java Enterprise Edition (Java Edición Empresarial) JEE versión 6 de la Oracle Corporation.

3.2.3 ENTERPRISE JAVA BEAN

Enterprise Java Beans es un modelo de componentes distribuido estándar del lado del servidor de la plataforma Java Edición Empresarial (JEE). La tecnología EJB permite el desarrollo rápido y simplificado de aplicaciones distribuidas, transaccionales, seguras y portátiles basadas en la tecnología Java[60].

59 **Ilustración obtenida de:** http://java.sun.com/developer/technicalArticles/J2EE/Intro/container1.jpg

60 **Información obtenida de:** http://java.sun.com/products/ejb/

Los EJB's son unas de las API's[61] que forman parte del estándar de construcción de aplicaciones empresariales J2EE (ahora JEE 6) de Oracle Corporation.

El objetivo de los EJB's es proporcionar a los desarrolladores un modelo que permita abstraerse de los problemas generales de la aplicaciones empresariales (concurrencia, transacciones, persistencia, seguridad, etc) para centrarse en el desarrollo de la lógica del negocio.

3.2.3.1 TIPOS DE ENTERPRISE JAVA BEANS

Hay tres tipos de EJB's:

3.2.3.1.1 EJB's DE ENTIDAD (ENTITY EJB's)

Encapsulan los objetos del lado del servidor que almacenan los datos. Los EJB de entidad presentan la característica fundamental de la persistencia:

Persistencia gestionada por el contenedor: es el encargado de almacenar y recuperar los datos a través del Mapeo Objeto Relacional.

Persistencia gestionada por el bean: el propio objeto se encarga de almacenar y recuperar los datos, por lo tanto el desarrollador es el encargo de implementar la persistencia.

61 **Interfaz de Programación de Aplicaciones** o API (Application Programming Interface) es el conjunto de funciones y métodos que ofrece una biblioteca para ser utilizada por otro software como una capa de abstracción.

3.2.3.1.2 EJB's DE SESIÓN (SESSION EJB's)

Gestionan el flujo de la información en el servidor, sirven a los clientes como acceso a los servicios proporcionados por los otros componentes que se encuentran el servidor.

Con sesión (Statefull): son objetos distribuidos que poseen un estado. El estado no es persistente, pero el acceso al bean se limita a un cliente.

Sin sesión (Stateless): son objetos distribuidos que carecen de un estado permitiendo por lo tanto el acceso concurrente.

3.2.3.1.3 EJB's DIRIGIDOS POR MENSAJE (MESSAGE-DRIVEN EJB's)

Beans con funcionamiento asíncrono. Utilizan el Sistema de Mensajes de Java (JMS), estos se suscriben a un tema o cola y se activan al recibir un mensaje dirigido a dicho tema o cola.

3.2.4 JAVA SERVER FACES

Es una tecnología para aplicaciones Java basada en web simplifica el desarrollo de interfaces de usuario en aplicaciones de la plataforma Java Edición Empresarial (JEE). JSF utiliza la tecnología JSP (Java Server Pages) para hacer el despliegue de las páginas.

Actualmente se ha difundido la integración de JSF con los clientes ricos de AJAX como son:

RICHFACES: Es una biblioteca de componentes para JSF y un marco avanzado de integración sencilla de capacidades AJAX en las aplicaciones empresariales[62].

Entre sus ventajas tenemos:

- 100 + AJAX habilitando los componentes de dos librerías:
  - a4j: Página de controles AJAX centralizada.
  - Rich: autónomo, listo para usar componentes.
- Conjunto de beneficios al trabajar con JSF y AJAX.
- Mecanismo de máscaras.
- Kit de desarrollo de componentes.
- Manejo de Recursos Dinámicos.
- Un amplio apoyo entre navegadores.
- Comunidad grande y activa.

ICEFACES: es más que una biblioteca de componentes de JSF y AJAX, es un marco de trabajo JEE AJAX para desarrollar y desplegar aplicaciones ricas empresariales, al igual que RichFaces; IceFaces comprende de similares ventajas[63].

62 **Referencia obtenida de:** http://www.jboss.org/richfaces

63 **Referencia obtenida de:** http://www.icefaces.org/

3.2.5 CONECTIVIDAD DE BASE DE DATOS JAVA

JDBC (Java DataBase Connectivity) es el estándar de la industria de la conectividad de base de datos independiente entre el lenguaje de programación Java y una amplia gama de bases de datos – bases de datos sql y otras fuentes de datos tabulares, tales como hojas de cálculo o ficheros planos. La API de JDBC proporciona una llamada de nivel de acceso basado en SQL de base de datos.

La tecnología JDBC permite utilizar el lenguaje de programación Java a explotar *"Write Once, Run Anywhere"*[64] capacidades para aplicaciones que requieren acceso a los datos de la empresa. Con la tecnología JDBC habilita y su controlador, puede conectar los datos de la empresa, incluso en un entorno heterogéneo[65].

64 **"Write Once, Run Anywhere"** es uno de las frases más utilizadas en programación significa "Escriba una vez, Ultilice cuando quiera".

65 **Información obtenida de:** http://java.sun.com/javase/technologies/database/index.jsp

3.2.6 SERVIDORES DE APLICACIONES JEE CERTIFICADOS

Los servidores de aplicaciones que se encuentran en las especificaciones de la plataforma Java Edición Empresarial (JEE) como certificados son:

SERVIDOR DE APLICACIONES	**EMPRESA**
GlassFish	Sun Microsystems
Apache Gerónimo	Apache Software Foundation
JBoss	RedHat
JOnAS	ObjectWeb
WebLogic Application Server	BEA Systems
SAP NetWeaver	SAP
JEUS	TMaxSoft
IBM WebShare Application Server	IBM
Oracle Containers	Oracle Corporation

Tabla 12: *Servidores de Aplicaciones JEE Certificados*

3.3 .NET (MICROSOFT)

3.3.1 INTRODUCCIÓN

La plataforma .NET de Microsoft es un componente de software que puede ser añadido al sistema operativo Windows. Provee un conjunto extenso de soluciones predefinidas para necesidades generales de la programación de aplicaciones, y administra la ejecución de los programas escritos específicamente con esta plataforma [WWW55].

Esta solución es el producto principal en la oferta de Microsoft, y pretende ser utilizada por la mayoría de aplicaciones creadas para la plataforma Windows .

3.3.2 DEFINICIÓN

"*.NET framework es la plataforma de desarrollo de código administrado por Microsoft. Esta formado por una serie de herramientas y librerías con las que se pueden crear todo tipo de aplicaciones, desde las tradicionales aplicaciones de escritorio (WPF o Windows Forms) hasta aplicaciones para XBOX (XNA) pasando por desarrollo web (ASP.NET), desarrollo para móviles (compact framework), aplicaciones de servidor (WPF, WCF), etcétera."*[66]

.NET es la respuesta de Microsoft al creciente avance en el mercado del desarrollo empresarial de JEE por parte de Oracle Corporation y de los diversos frameworks de desarrollo web basados en PHP.

66 **Información obtenida de:** http://msdn.microsoft.com/es-ec/netframework/default.aspx

3.3.3 COMPONENTES

Los principales componentes de Microsoft .NET son:

- Conjunto de lenguajes de programación.
- Entorno Común de Ejecución para Lenguajes (CRL).
- Biblioteca de Clases Base (BCL).

3.3.3.1 ENTORNO COMÚN DE EJECUCIÓN (CRL)

Ilustración 24: *Diagrama interno del Entorno Común de Ejecución para Lenguajes (CLR)*

.NET proporciona un entorno en tiempo de ejecución denominado *Entorno Común de Ejecución,* que ejecuta el código y proporciona servicios que facilitan el proceso de desarrollo como se muestra en la ilustración número 24[67].

67 **Ilustración tomada de:** http://es.wikipedia.org/wiki/Archivo:Diagrama_Interno_CLR.jpg

La herramienta de desarrollo compila el código fuente de cualquiera de los lenguajes soportados por el .NET framework en un código intermedio, el MSIL (Lenguaje Intermedio de Microsoft), similar al BYTECODE de Java. Para generarlo, el compilador se basa en la especificación CLS (Especificación de Lenguaje Común) que determina las reglas necesarias para crear el código del Lenguaje Intermedio de Microsoft compatible con el Entorno Común de Ejecución.

Posteriormente, necesita un compilador JIT (Just-In-Time) el cual es el encargado de generar el código de máquina para su ejecución, esta compilación es realiza por el Entorno Común de Ejecución a medida que se que se invocan los métodos [WWW55].

3.3.3.2 BIBLIOTECA DE CLASES BASE (BCL)

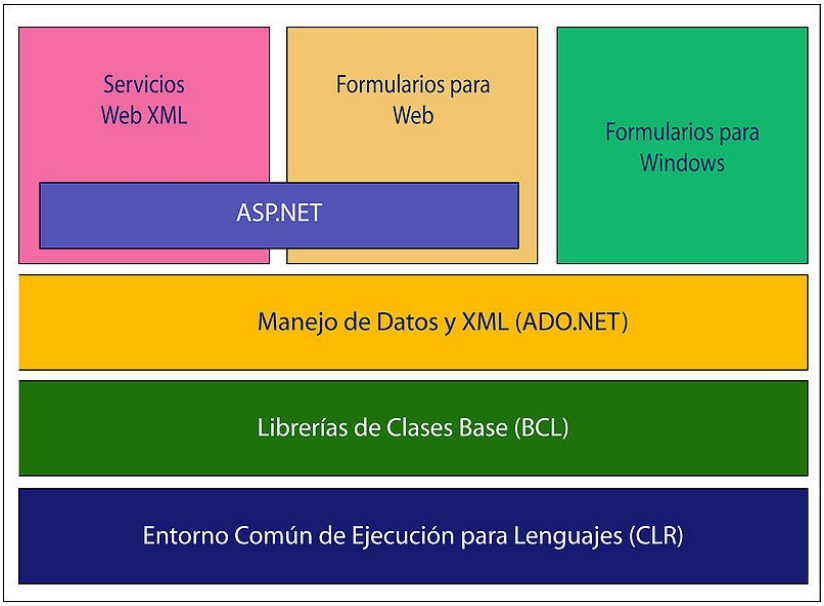


Ilustración 25: *Diagrama básico de la Biblioteca de Clases Base (BCL)*

.NET incluye clases, interfaces y tipos de valor que aceleran y optimizan el proceso de desarrollo y proporcionan el acceso a la funcionalidad del sistema. Para esto utilizan la especificación CLS (Especificación de Lenguaje Común), por lo tanto se puede utilizar en todo el conjunto de lenguajes de .NET como se muestra en la ilustración número 24[68].

68 **Ilustración tomada de:** http://es.wikipedia.org/wiki/Archivo:Diagrama_CLR.jpg

3.4 PHP

3.4.1 INTRODUCCIÓN

PHP es un acrónimo recursivo que significa PHP Hypertext Pre-processor, inicialmente fue creado como PHP Tools (Personal Home Page Tools). Fue inicialmente desarrollado por Rasmus Lerdorf[69] en 1994 actualmente es producida por la comunidad a través de "The PHP Group" con una licencia PHP Licence que es una licencia de software libre.

3.4.2 DEFINICIÓN

"PHP es un lenguaje interpretado de propósito general ampliamente utilizado, diseñado especialmente para desarrollo web y que puede ser incrustado dentro de código HTML.[70]"

La actual versión de PHP 5 incluye las siguientes mejoras:

- Mejor soporte para Programación Orientada a Objetos con PDO[71].
- Mejoras en el rendimiento.

69 R**asmus Lerdorf**, es un programador informático nacido en Groelandia, es considerado el autor principal de PHP.

70 **Información obtenida de:** http://www.php.net/

71 **PDO (PHP Data Objects)**, según wikipedia es una extensión que provee una capa de abstracción de acceso a datos para PHP 5, con lo cual se consigue hacer uso de las mismas funciones para hacer consultas y obtener datos de distintos manejadores de bases de datos.

- Mejor soporte para MySql con extensión completamente reescrita.
- Mejor soporte a XML (Xpath, DOM, etc).
- Soporte nativo para SQLite.
- Soporte integrado para SOAP[72].
- Iteradores de datos.
- Manejo de excepciones.
- Mejoras con la implementación de Oracle.

En la ilustración número 26 se muestra la estructura tecnológica de PHP.

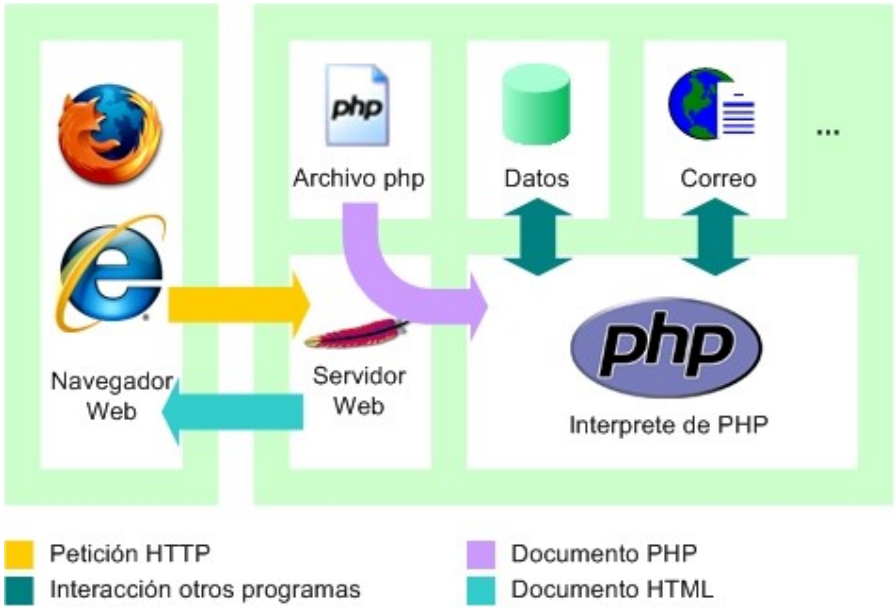


Ilustración 26: *Arquitectura Tecnológica PHP*

72 **SOAP (Simple Object Access Protocol)** Protocolo de Acceso Simple a Objetos es un protocolo estándar que define como objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML [WWW56]

3.4.3 PEAR

Según el sitio web oficial de PHP *"PEAR (PHP Extensión y Repositorios de Aplicación) es un entorno de desarrollo y sistema de distribución de componentes de PHP"*.

3.4.4 FRAMEWORKS

A continuación las características de dos de los frameworks de PHP más utilizados:

ZEND FRAMEWORK

Zend Framework no requiere instalación especial, solo necesita PHP 5 e incorpora el patrón MVC (Modelo Vista Controlador).

SYMFONY:

Fue diseñado con el objetivo de crear aplicaciones web, con el uso de sus características. Posee una librería de clases que permite reducir el tiempo de desarrollo.

Symfony esta desarrollado en PHP 5, requiere de instalación, configuración y líneas de comando incorpora el patrón MVC (Modelo Vista Controlador), soporta AJAX, plantillas y un gran número de motores de base de datos.

3.5 COMPARACIÓN DE ARQUITECTURAS TECNOLÓGICAS

	JEE	**.NET**	**PHP**
Lenguajes de programación	Java.	C#, Visual Basic, Visual C++, J#.	PHP.
Influencia de lenguajes	Objetive-C, C++, Smalltalk, Eiffel.	Java, C++, C, Pascal, Eiffel.	C, C++, Perl, Java y Python.
Sistemas Operativos	Linux, Windows, Unix, BSD, Mac	Windows	Linux, Windows, Unix, BSD, Mac
Licencia de Software	GNU GPL/Java (Software Libre).	Comercial	PHP Licence 3.0.1 (Software Libre)

Tabla 13: *Tabla comparativa de Arquitecturas Tecnológicas*

En el mundo de Java la plataforma es denominada actualmente JEE, sin embargo a diferencia de .NET que es un concepto global, JEE es un grupo de especificaciones, la principal ventaja que JEE este basado en especificaciones es la *libertad de elección* sobre proveedores, esto es, los componentes escritos en Java son interoperables entre productos JEE desarrollados por IBM, HP, Oracle Sun, BEA, etc; a diferencia de .NET donde todo gira alrededor de un sólo proveedor: Microsoft; la diferencia entre PHP y .NET es similar a la ya expuesta con JEE.

En cambio entre JEE y PHP la diferencia radica en el tamaño de nuestros proyectos informáticos JEE esta orientado a grandes proyectos y PHP a pequeños.

CAPÍTULO IV

4 APLICATIVO

4.1 INTRODUCCIÓN

En la sección de ***Gestión del Proyecto*** se muestran las planificaciones de desarrollo del proyecto, así como el cronograma de ejecución del proyecto, de construcción de la aplicación y cumplimiento de los plazos estimados.

En la sección de ***Modelado del Negocio*** se encuentran los artefactos utilizados de la metodología RUP para definir un modelo del negocio, modelos de objetos del negocio y el modelo del dominio.

En la sección ***Requisitos*** se encuentra los artefactos definidos según la metodología RUP, es decir, el documento plan de desarrollo de software, el documento visión, el documento glosario, matrices de atributos de todos los requerimientos, los casos de uso y sus especificaciones.

En la sección ***Análisis/Diseño*** se muestran tanto el modelo de análisis/diseño (diagrama de clases) como el modelo de datos (modelo entidad – relación).

En la sección ***Implementación*** se muestran los prototipos de interfaces de usuario de la aplicación.

Por último, en el apartado ***Pruebas*** se encuentran las especificaciones de casos de pruebas funcionales.

A continuación se indican las herramientas y tecnologías que se utilizaron en el desarrollo del sistema:

ENTORNO DE DESARROLLO:

Hardware:

- **Marca:** Dell
- **Modelo:** Vostro 1400
- **Procesador:** Intel Core 2 Duo 2.0 Ghz
- **Memoria RAM:** 4Gb
- **Disco Duro:** 250Gb

Software:

- **Sistema Operativo:** Open SuSe 11.2
- **Lenguaje de programación:** Java
- **Sistema de Gestión de Base de Datos Relacional:** Postgresql 8.4.3
- **Entorno de desarrollo:** NetBeans 6.9
- **Servidor de aplicaciones:** GlassFish 3.1
- **Framework:** Spring 3.0.2
- **Cliente Rico:** RichFaces (JSF 2.0)
- **Modelador UML:** ArgoUML 0.28.1
- **Suite Ofimática:** OpenOffice 3.2

ENTORNO DE PRODUCCIÓN:

Hardware:

- **Marca:** HP
- **Modelo:** ProiLant ML115
- **Procesador:** Intel Xeon Core 2 Duo
- **Memoria RAM:** 2Gb
- **Disco Duro:** 500Gb

Software:

- **Sistema Operativo:** SuSe Linux Enterprise Server 11
- **Sistema de Gestión de Base de Datos:** Postgresql 8.4.3
- **Servidor de Aplicaciones:** GlassFish 3.1

4.2 GESTIÓN DEL PROYECTO

En esta sección se detalla la planificación inicial del proyecto para la fase de inicio y la fase de elaboración (según la definición de la metodología RUP).

4.2.1 PLAN DE DESARROLLO DE SOFTWARE

4.2.1.1 INTRODUCCIÓN

Este Plan de Desarrollo de Software es una versión preliminar preparada para ser incluida en la propuesta elaborada como respuesta al trabajo final de grado previo a la obtención del título de Ingeniero en Sistemas Computacionales de la Facultad de Ingeniería en Ciencias Aplicadas de la Universidad Técnica del Norte.

El enfoque de desarrollo propuesto constituye una configuración del proceso RUP de acuerdo a las características del proyecto, seleccionando los roles de los participantes, las actividades a realizar y los artefactos (entregables) que serán generados. Este documento es a su vez uno de los artefactos de RUP.

4.2.1.1.1 PROPÓSITO

El propósito del Plan de Desarrollo de Software es proporcionar la información necesaria para controlar el proyecto. En él se describe el enfoque de desarrollo de software.

Los usuarios del Plan de Desarrollo de Software son:

- El jefe de proyecto lo utiliza para organizar la agenda y necesidades de recursos, y para realizar su seguimiento.

- Los miembros del equipo de desarrollo lo usan para entender lo qué deben hacer, cuándo deben hacerlo y que otras actividades dependen de ello.

4.2.1.1.2 ALCANCE

El Plan de Desarrollo de Software describe el plan global usado para el desarrollo del *"Sistema de Planificación de Recursos Empresariales" – FinanSoft.*

Durante el proceso de desarrollo en el artefacto "Visión" se definen las características del producto a desarrollar, lo cual constituye la base para la planificación de las iteraciones.

Para la versión 1.0 del Plan de Desarrollo de Software nos hemos basado en la captura de requisitos por medio del stakeholder representante de la empresa para hacer una estimación aproximada, una vez comenzado el proyecto y durante la fase de Inicio se generará la primera versión del artefacto "Visión", el cual se utilizará para refinar este documento.

Posteriormente, el avance del proyecto y el seguimiento en cada una de las iteraciones ocasionará el ajuste de este documento produciendo nuevas versiones actualizadas.

4.2.1.1.3 RESUMEN

Después de esta introducción, el resto del documento está organizado en las siguientes secciones:

Vista General del Proyecto – proporciona una descripción del propósito, alcance y objetivos del proyecto, estableciendo los artefactos que serán producidos y utilizados durante el proyecto.

Organización del Proyecto – describe la estructura organizacional del equipo de desarrollo.

Gestión del Proceso – explica los costos y planificación estimada, define las fases e hitos del proyecto y describe cómo se realizará su seguimiento.

Planes y Guías de aplicación – proporciona una vista global del proceso de desarrollo de software, incluyendo métodos, herramientas y técnicas que serán utilizadas.

4.2.1.2 VISTA GENERAL DEL PROYECTO

4.2.1.2.1 PROPÓSITO, ALCANCE Y OBJETIVOS

La información que a continuación se incluye a sido extraída de las diferentes reuniones que se han celebrado con el stakeholder de la empresa desde el inicio del proyecto.

El proyecto debe proporcionar una respuesta para el desarrollo de todos los módulos implicados en el *"Sistema de Planificación de Recursos Empresariales" – FinanSoft.* Estos módulos son los siguientes:

- **CONTABILIDAD**
  - Plan de Cuentas.
  - Jornalización
  - Mayorización.
  - Balances.
  - Estados Financieros

- **FACTURACIÓN**
  - Clientes.
  - Ventas.

- **INVENTARIO**
  - Productos.
  - Proveedores.
  - Compras.

- **RECURSOS HUMANOS**
  - Inventario de personal.

- **SEGURIDAD**
  - Usuarios.
  - Roles.
  - Menús.

4.2.1.2.2 SUPOSICIONES Y RESTRICCIONES

Las suposiciones y restricciones respecto del sistema, y que se derivan directamente de las entrevistas con el stakeholder de la empresa son:

a) Debe contemplarse las implicaciones de los siguientes puntos críticos:
   a.a) Sistemas seguros: protección de información, seguridad en las transmisiones de datos, etc.
   a.b) Gestión de flujos de trabajo, seguridad de las transacciones e intercambio de información.

b) La automatización de la gestión interna del registro debe ajustarse a la legislación vigente.

c) El módulo de facturación debe ser desarrollado como un sistema independiente en aplicación de escritorio para ser utilizado por todas las sucursales de la empresa.

Como es natural, la lista de suposiciones y restricciones se incrementará durante el desarrollo del proyecto, particularmente una vez establecido el artefacto “Visión”.

4.2.1.2.3 ENTREGABLES DEL PROYECTO

A continuación se indican y describen cada uno de los artefactos que serán generados y utilizados por el proyecto y que constituyen los entregables. Esta lista constituyenla configuración de RUP desde la perspectiva de artefactos, y que proponemos para este proyecto.

Es preciso destacar que de acuerdo a la filosofía de RUP (y de todo proceso iterativo e incremental), todos los artefactos son objetos de modificaciones a lo largo del proceso de desarrollo, con lo cual, solo al término del proceso podríamos tener una versión definitiva y completa de cada uno de ellos.

Sin embargo, el resultado de cada iteración y los hitos del proyecto están enfocados a conseguir un cierto grado de complejidad y estabilidad de los artefactos. Esto será indicado más adelante cuando se presenten los objetivos de cada iteración.

Plan de Desarrollo de Software

Es el presente documento.

Modelo de Casos de Uso del Negocio

Es un modelo de las funciones de negocio vistas desde la perspectiva de los actores externos (Agentes de registro, solicitantes finales, otros sistemas, etc.) permite situar al sistema en el contexto organizacional haciendo énfasis en los objetivos en este ámbito. Este modelo se representa con un Diagrama de Casos de Uso usando estereotipos específicos para este modelo.

Modelo de Objetos del Negocio

Es un modelo que describe la realización de cada caso del negocio, estableciendo los actores internos, la información que en términos generales manipulan y los flujos de trabajo asociados al caso de uso del negocio.

Para la representación de este modelo se utilizan Diagramas de Colaboración (para mostrar actores externos, internos y las entidades que manipulan), un Diagrama de Clases para mostrar gráficamente las entidades del sistema y sus relaciones, y un Diagrama de Actividades para mostrar los flujos de trabajo.

Glosario

Es un documento que define los principales términos usados en el proyecto.

Modelo de Casos de Uso

El modelo de Casos de Uso presenta las funciones del sistema y los actores que hacen uso de ellas, con Diagramas de Casos de Uso.

Visión

Este documento define la visión del producto desde la perspectiva del cliente, especificando las necesidades y características del producto. Constituye una base de acuerdo a los requisitos del sistema.

Especificaciones de Casos de Uso

Para los casos de uso que se requieran (cuya funcionalidad no sea evidente o que no baste con una simple descripción narrativa) se realiza una descripción detallada utilizando una plantilla de documento, donde se incluyen: pre – condiciones, post – condiciones, flujos de eventos, requisitos no – funcionales asociados. También para casos de uso cuyo flujo de eventos sea complejo podrá adjuntarse una representación gráfica mediante un Diagrama de Actividad.

Especificaciones Adicionales

Este documento capturará todos los requisitos que no han sido incluidos como parte de los casos de uso y se refieren a requisitos no – funcionales globales. Dichos requisitos incluyen: requisitos legales o normas, aplicación de estándares, requisitos de calidad del producto, tales como: confiabilidad, desempeño, etc., u otros requisitos de ambiente, tales como: sistema operativo, requisitos de compatibilidad, etc.

Prototipos de Interfaces de Usuario

Se trata de prototipos que permiten al usuario hacerse una idea más o menos precisa de las interfaces que proveerá el sistema, y así conseguir una retroalimentación de su parte respecto a los requisitos del sistema. Estos prototipos se realizarán como: dibujos a mano en papel, dibujos con alguna herramienta gráfica o prototipos ejecutables interactivos, siguiendo ese orden de acuerdo al avance del proyecto.

Sólo los de este último tipo serán entregados al final de esta fase de Elaboración, los otros serán desechados. Así mismo, este artefacto, será desechado en la fase final de Construcción en la medida que el resultado de las iteraciones vayan desarrollando el producto final.

Modelo de Análisis y Diseño

Este modelo establece la realización de casos de uso en clases y pasando desde una representación en términos de análisis (sin incluir aspectos de implementación) hacia una de diseño (incluyendo una orientación hacia el entorno de implementación), de acuerdo al avance del proyecto.

Modelo de Datos

Previendo que la persistencia de la información del sistema será soportada por una base de datos relacional, este modelo describe la presentación lógica de los datos persistentes, de acuerdo con el enfoque para modelado relacional de daos. Para expresar este modelo se utiliza un Diagrama de Clases.

Modelo de Despliegue

Este modelo muestra el despliegue la configuración de tipos de nodos del sistema, en los cuales se hará el despliegue de los componentes.

Modelo de Implementación

Este modelo es una colección de componentes y los módulos que los contienen. Estos componentes incluyen: ficheros ejecutables, ficheros de código fuente, y todo otro tipo de ficheros necesarios para la implantación y despliegue del sistema.

Casos de Prueba

Cada prueba es especificada mediante un documento que establece las condiciones de ejecución, las entradas de la prueba, y los resultados esperados. Estos casos de prueba son aplicados como pruebas de regresión en cada iteración. Cada caso de prueba llevará asociado un procedimiento de prueba con las instrucciones para realizar la prueba.

Solicitud de Cambio

Los cambios propuestos para los artefactos se formalizarán mediante este documento. Mediante este documento se hace un seguimiento de los defectos, solicitud de mejoras o cambios en los requisitos del producto. Así se provee un registro de decisiones de cambios, de su evaluación e impacto, y se asegura que éstos sean conocidos por el equipo de desarrollo.

Los cambios se establecen respecto de la última baseline (el estado del conjunto de los artefactos en un momento determinado del proyecto) establecida. En nuestro caso al final de cada iteración se establecerá una baseline.

Plan de Iteración

Es un conjunto de actividades y tareas ordenadas temporalmente, con recursos asignados, dependencias entre ellas.

Evaluación de Iteración

Este documento incluye la evaluación de los resultados de cada iteración, el grado en el cual se han conseguido los objetivos de la iteración, las lecciones aprendidas y los cambios a ser realizados.

Lista de Riesgos

Este documento incluye una lista de los riesgos conocidos y vigentes en el proyecto, ordenados en orden decreciente de importancia y con acciones específicas de contingencia o para su mitigación.

Manual de Instalación

Este documento incluye las instrucciones para realizar la instalación del producto.

Material de Apoyo al Usuario Final

Corresponde a un conjunto de documentos y facilidades de uso del sistema, incluyendo: Guías del Usuario, Guías de Operación, Guías de Mantenimiento y Sistema de Ayuda en línea.

Producto

Los ficheros del producto empaquetados y almacenados en un medio digital con los mecanismos apropiados para facilitar su instalación. El producto, a partir de la primera iteración de la fase de Construcción es desarrollado incremental e iterativamente, obteniéndose una nueva release al final de cada iteración.

4.2.1.2.4 EVOLUCIÓN DEL PLAN DE DESARROLLO DE SOFTWARE

El Plan de Desarrollo de Software se revisará y refinará al comienzo de cada iteración.

4.2.1.3 ORGANIZACIÓN DEL PROYECTO

4.2.1.3.1 PARTICIPANTES EN EL PROYECTO

El personal participante en el proyecto esta formado por los siguientes puestos de trabajo y personal asociado:

Jefe de Proyecto: Ing. Irving Reascos.
Arquitecto de Software: Egdo. Alcides Rivera Posso.
Ingeniero de Software: Egdo. Alcides Rivera Posso.
Programador: Egdo. Alcides Rivera Posso.

4.2.1.3.2 INTERFACES EXTERNAS

La empresa definirá los participantes del proyecto que proporcionarán los requisitos del sistema, y entre ellos quiénes serán los encargados de evaluar los artefactos de acuerdo a cada módulo y según el plan establecido.

El equipo de desarrollo interactuará activamente con los participantes de la empresa para especificación y validación de los artefactos generados.

4.2.1.3.3 ROLES Y RESPONSABILIDADES

A continuación se describen las principales responsabilidades de cada uno de los puestos en el equipo de desarrollo, de acuerdo con los roles que desempeñan en RUP.

Puesto	Responsabilidad
Jefe de Proyecto	El jefe de proyecto asigna los recursos, gestiona las prioridades, coordina las interacciones con los clientes y usuarios, y mantiene al equipo del proyecto enfocado en los objetivos. El jefe de proyecto también establece un conjunto de prácticas que aseguran la integridad y calidad de los artefactos del proyecto. Gestión de riesgos, Planificación y control del proyecto.
Arquitecto de Software	Se encargará de supervisar el establecimiento de la Arquitectura del Sistema, es decir, definir la vista arquitectónica, los estilos arquitectónicos, el patrón de arquitectura y la arquitectura tecnológica a utilizar. Establecer la conectividad entre las diferentes sucursales de la empresa.
Ingeniero de Software	Gestión de requisitos, gestión de configuración y cambios, elaboración del modelo de datos, preparación de las pruebas funcionales, elaboración de la documentación. Elaborar modelos de implementación y despliegue. Captura, especificación y validación de requisitos, interactuando con el cliente y los usuarios mediante entrevistas. Elaboración del Modelo de Análisis y Diseño. Colaboración en la elaboración de las pruebas funcionales y el modelo de datos. Encargado además de la puesta en producción de la empresa.
Programador	Construcción de prototipos. Colaboración en la elaboración de las pruebas funcionales, modelo de datos y en las validaciones con el usuario.
Tester	Se encargará de la realización de las pruebas funcionales, de conectividad y rendimiento del sistema.

4.2.1.4 GESTIÓN DEL PROCESO

4.2.1.4.1 ESTIMACIONES DEL PROYECTO

El presupuesto del proyecto y los recursos involucrados se adjuntan en un documento por separado.

4.2.1.4.2 PLAN DEL PROYECTO

En esta sección se presenta la organización en fases e iteraciones y el calendario del proyecto.

4.2.1.4.2.1 PLAN DE FASES

El desarrollo se llevará a cabo en base a fases con una o más iteraciones en cada una de ellas. La siguiente tabla muestra la distribución de tiempos y el número de iteraciones de cada fase.

Fase	Nro. Iteraciones	Duración
Fase de Inicio	1	3 semanas
Fase de Elaboración	2	2 semanas
Fase de Construcción	3	7 semanas
Fase de Transición	2	2 semanas

Tabla 14: *Plan de Fases del Proyecto*

Los hitos que marcan el final de cada fase se describen en la siguiente tabla:

Descripción	Hito
Fase de Inicio	En esta fase se desarrollará los requisitos del producto desde la perspectiva del usuario, los cuales serán establecidos en el artefacto Visión. Los principales casos de uso serán identificados y se hará un refinamiento al Plan de Desarrollo de Software. La aceptación del cliente/usuario del artefacto Visión y el Plan de Desarrollo de Software marcan el final de esta fase.
Fase de Elaboración	En esta fase se analizan los requisitos y se desarrolla un prototipo de arquitectura (incluyendo las partes mas relevantes y/o críticas del sistema). Al final de esta fase, todos los casos de uso correspondientes a requisitos que serán implementados en la primera release de la fase de Construcción deben ser analizados y diseñados (en el Modelo de Análisis/Diseño). La revisión y aceptación del prototipo de la arquitectura del sistema marca el final de esta fase.
Fase de Construcción	Durante la fase de construcción se terminan de analizar y diseñar todos los casos de uso, refinando el Modelo de Análisis/Diseño. El producto se construye en base a dos iteraciones, cada una produciendo un release a la cual se le aplican las pruebas y se valida con el cliente/usuario. Se comienza la elaboración del material de apoyo al usuario. El hito que marca el fin de esta fase es la versión de la release 3.0, con la capacidad operacional parcial del producto que se haya considerado como crítica, lista para ser entregada a los usuarios para la prueba beta.
Fase de Transición	En esta fase prepararán dos release para distribución, asegurando una implantación y cambio del sistema previo de manera adecuada, incluyendo el entrenamiento de los usuarios. El hito que marca el fin de esta fase incluye, la entrega de toda la documentación del proyecto con los manuales de instalación y todo el material de apoyo al usuario, la finalización del entrenamiento de los usuarios y el empaquetado del producto, además se establecerá la conectividad entre las diferentes sucursales de la empresa.

Tabla 15: *Hitos de las fases del Proyecto*

4.2.1.4.2.2 CALENDARIO DEL PROYECTO

A continuación se presenta un calendario de las principales tareas del proyecto. Como se ha comentado, el proceso iterativo e incremental de RUP esta caracterizado por la realización en paralelo de todas las disciplinas de desarrollo a lo largo del proyecto, con lo cual la mayoría de los artefactos son generados muy tempranamente en el proyecto pero van desarrollándose en mayor o menor grado de acuerdo a la fase e iteración del proyecto.

La siguiente figura ilustra este enfoque, en ella lo ensombrecido marca el énfasis de cada disciplina (workflow) en un momento determinado del desarrollo.

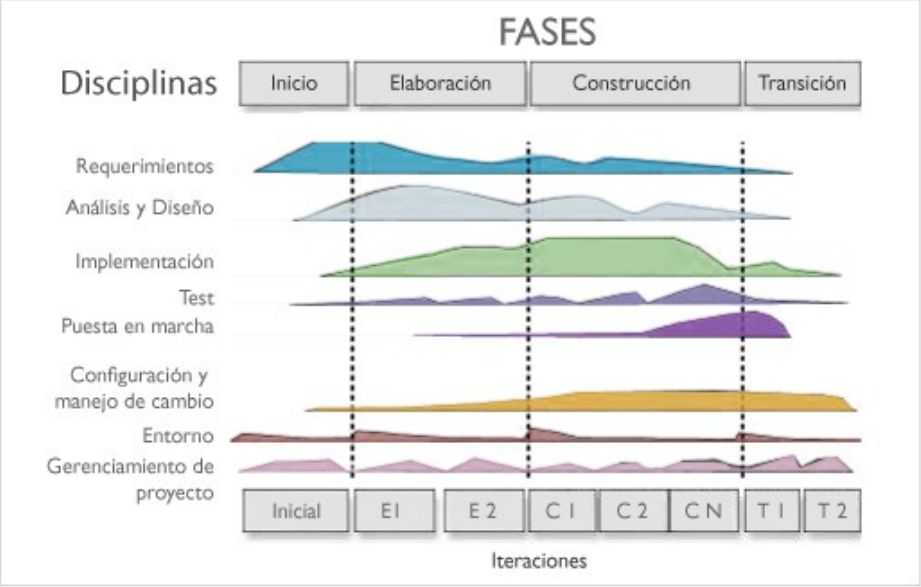


Ilustración 27: *Iteraciones del Proyecto*

Para este proyecto se ha establecido el siguiente calendario. La fecha de aprobación indica cuándo el artefacto en cuestión tiene un estado de completitud suficiente para someterse a revisión y aprobación, pero eso no quita la posibilidad de su posterior refinamiento y cambios.

Disciplinas/Artefactos generados o modficados durante la Fase de Inicio	**Comienzo**	**Aprobación**
Modelado del Negocio		
Modelo de Casos de Uso del Negocio y Modelado de Objetos del Negocio	Semana 03/10/09 – 07/10/09	Semana 17/10/09 – 21/10/09
Requisitos		
Glosario	Semana 03/10/09 – 07/10/09	Semana 17/10/09 – 21/10/09
Visión	Semana 10/10/09 – 14/10/09	Semana 17/10/09 – 21/10/09
Modelo de Casos de Uso	Semana 17/10/09 – 21/10/09	Siguiente fase
Especificación de Casos de Uso	Semana 17/10/09 – 21/10/09	Siguiente fase
Especificaciones Adicionales	Semana 17/10/09 – 21/10/09	Siguiente fase
Análisis/Diseño		
Modelo de Análisis/Diseño	Semana 10/10/09 – 14/10/09	Siguiente fase
Modelo de Datos	Semana 10/10/09 – 14/10/09	Siguiente fase
Implementación		
Prototipos de Interfaces de Usuario	Semana 17/10/09 – 21/10/09	Siguiente fase
Modelo de Implementación	Semana 17/10/09 – 21/10/09	Siguiente fase

Pruebas		
Casos de Prueba Funcionales	Semana 17/10/09 – 21/10/09	Siguiente fase
Despliegue		
Modelo de Despliegue	Semana 17/10/09 – 21/10/09	Siguiente fase
Gestión de Cambios y Configuración	Durante todo el proyecto	Durante todo el proyecto
Gestión del Proyecto		
Plan de Desarrollo de Software en su versión 1.0 y planes de las Iteraciones	Semana 03/10/09 – 07/10/09	Semana 17/10/09 – 21/10/09
Ambiente	Durante todo el proyecto	Durante todo el proyecto

Tabla 16: *Calendario del Proyecto: Fase de Inicio*

Disciplinas/Artefactos generados o modficados durante la Fase de Elaboración	**Comienzo**	**Aprobación**
Modelado del Negocio		
Modelo de Casos de Uso del Negocio y Modelado de Objetos del Negocio	Semana 03/10/09 – 07/10/09	Aprobado
Requisitos		
Glosario	Semana 03/10/09 – 07/10/09	Aprobado
Visión	Semana 10/10/09 – 14/10/09	Aprobado
Modelo de Casos de Uso	Semana 17/10/09 – 21/10/09	Semana 05/10/09 - 09/10/09
Especificación de Casos de Uso	Semana 17/10/09 – 21/10/09	Semana 05/10/09 - 09/10/09
Especificaciones Adicionales	Semana 17/10/09 – 21/10/09	Semana 05/10/09 - 09/10/09

Análisis/Diseño		
Modelo de Análisis/Diseño	Semana 10/10/09 – 14/10/09	Revisar en cada iteración
Modelo de Datos	Semana 10/10/09 – 14/10/09	Revisar en cada iteración
Implementación		
Prototipos de Interfaces de Usuario	Semana 17/10/09 – 21/10/09	Revisar en cada iteración
Modelo de Implementación	Semana 17/10/09 – 21/10/09	Revisar en cada iteración
Pruebas		
Casos de Prueba Funcionales	Semana 17/10/09 – 21/10/09	Revisar en cada iteración
Despliegue		
Modelo de Despliegue	Semana 17/10/09 – 21/10/09	Revisar en cada iteración
Gestión de Cambios y Configuración	Durante todo el proyecto	Durante todo el proyecto
Gestión del Proyecto		
Plan de Desarrollo de Software en su versión 1.0 y planes de las Iteraciones	Semana 03/10/09 – 07/10/09	Revisar en cada iteración
Ambiente	Durante todo el proyecto	Durante todo el proyecto

Tabla 17: *Calendario del Proyecto: Fase de Elaboración*

Disciplinas/Artefactos generados o modificados durante la Fase de Construcción (Iteración 1)	**Comienzo**	**Aprobación**
Casos de Uso negociados para la Primera Release		
Gestión del Plan de Cuentas	19/11/09	21/12/09
Buscar Cuenta	19/11/09	21/12/09
Gestión de Clientes	23/12/09	21/12/09
Buscar Cliente	23/12/09	21/12/09

Gestión de Proveedores	27/11/09	21/12/09
Buscar Proveedor	27/11/09	21/12/09
Gestión de Productos	01/12/09	21/12/09
Buscar Producto	01/12/09	21/12/09
Gestión de Empleados	05/12/09	21/12/09
Buscar Empleado	05/12/09	21/12/09
Manejo de Usuarios	09/12/09	21/12/09
Buscar Usuario	09/12/09	21/12/09
Manejo de Roles	13/12/09	21/12/09
Buscar Rol	13/12/09	21/12/09
Manejo de Menús	17/12/09	21/12/09
Buscar Menú	17/12/09	21/12/09

Tabla 18: *Calendario del Proyecto: Fase de Construcción (Iteración 1)*

Disciplinas/Artefactos generados o modificados durante la Fase de Construcción (Iteración 2)	**Comienzo**	**Aprobación**
Casos de Uso negociados para la Primera Release		
Gestión del Plan de Cuentas	19/11/09	Aprobado
Buscar Cuenta	19/11/09	Aprobado
Gestión de Clientes	23/12/09	Aprobado
Buscar Cliente	23/12/09	Aprobado
Gestión de Proveedores	27/11/09	Aprobado
Buscar Proveedor	27/11/09	Aprobado
Gestión de Productos	01/12/09	Aprobado
Buscar Producto	01/12/09	Aprobado
Gestión de Empleados	05/12/09	Aprobado
Buscar Empleado	05/12/09	Aprobado
Manejo de Usuarios	09/12/09	Aprobado

Buscar Usuario	09/12/09	Aprobado
Manejo de Roles	13/12/09	Aprobado
Buscar Rol	13/12/09	Aprobado
Manejo de Menús	17/12/09	Aprobado
Buscar Menú	17/12/09	Aprobado
Casos de Uso negociados para la Segunda Release		
Gestión de Ventas	13/01/10	02/02/10
Buscar Venta	13/01/10	02/02/10
Gestión de Compras	17/01/10	02/02/10
Buscar Compra	17/01/10	02/02/10
Login – Logout	21/01/10	02/02/10
Gestión del Libro Diario	25/01/10	02/02/10
Buscar Asiento	25/01/10	02/02/10
Gestionar Reportes	29/01/10	02/02/10

Tabla 19: *Calendario del Proyecto: Fase de Construcción (Iteración 2)*

4.2.1.4.2.3 SEGUIMIENTO Y CONTROL DEL PROYECTO

Gestión de Requisitos

Los requisitos del sistema son especificados en el artefacto Visión. Cada requisito tendrá una serie de atributos tales como importancia, estado, iteración, donde se implementa, etc. Estos atributos permitirán realizar un efectivo seguimiento de cada requisito. Los cambios en los requisitos serán gestionados mediante una Solicitud de Cambio, las cuales serán evaluadas y distribuidas para asegurar la integridad del sistema y el correcto proceso de gestión y configuración de cambios.

Control de Plazos

El calendario del proyecto tendrá un seguimiento y evaluación semanal por el jefe de proyecto.

Control de Calidad

Los defectos detectados en las revisiones y formalizados también en una Solicitud de Cambio tendrán un seguimiento para asegurar la conformidad al respecto de la solución de dichas deficiencias. Para la revisión de cada artefacto y su correspondiente garantía se utilizará las guías de revisión y checklist (listas de verificación) incluidas en RUP.

Gestión de Riesgos

A partir de la fase de Inicio se mantendrá una lista de riesgos asociados al proyecto y de las acciones establecidas como estrategia para mitigarlos o acciones de contingencia. Esta lista será evaluada al menos una vez cada iteración.

Gestión de Configuración

Se realizará una gestión de configuración para llevar un registro de los artefactos generados y sus versiones. También se incluirá la gestión de las Solicitudes de Cambio y de las modificaciones que éstas produzcan, informando y publicando dichos cambios para que sean accesibles a todos los participantes en el proyecto.

Al final de cada iteración se establecerá un baseline (un registro del estado de cada artefacto, estableciendo una versión), la cual podrá ser modificada solo por una Solicitud de Cambio aprobada.

4.3 MODELADO DEL NEGOCIO

A continuación se presentan los modelos definidos en RUP como modelo del negocio, modelo de datos y modelo de análisis y diseño.

CONTABILIDAD:

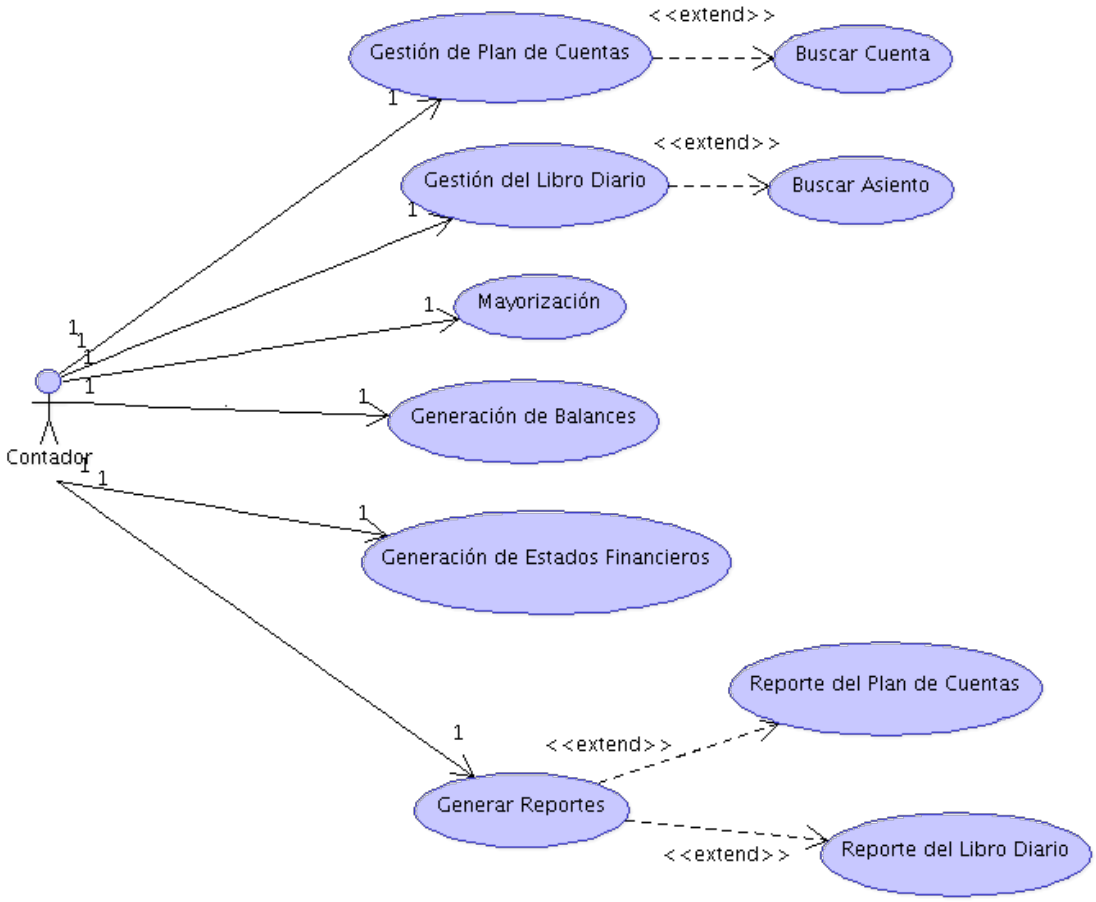


Ilustración 28: *Diagrama de Caso de Uso del Módulo de Contabilidad*

FACTURACIÓN:

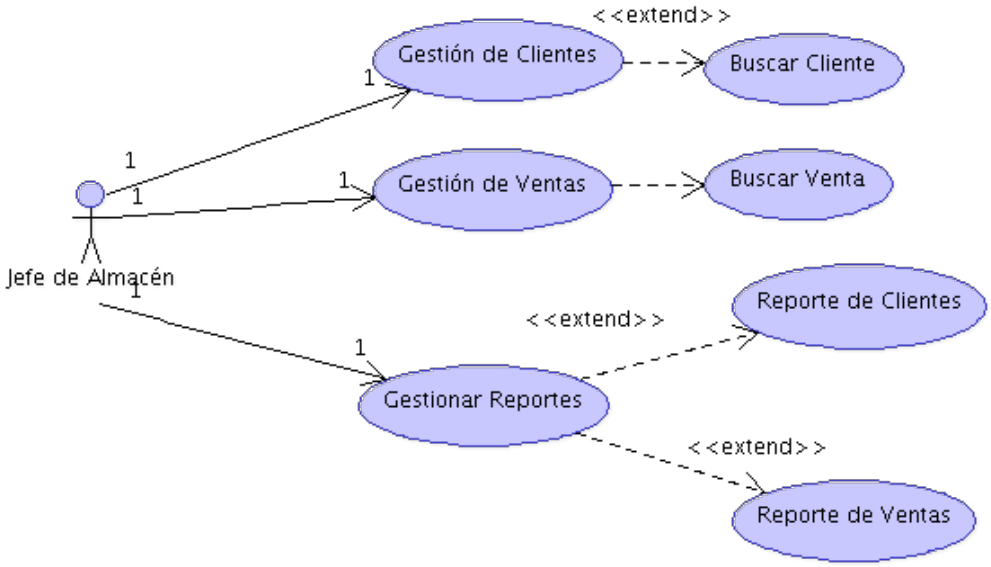


Ilustración 29: *Diagrama de Caso de Uso del Módulo de Facturación*

GENERAL:

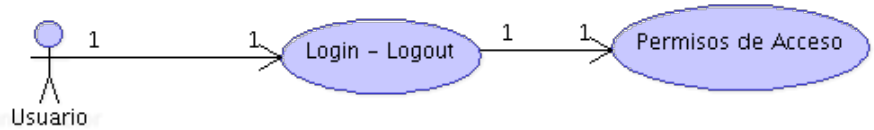


Ilustración 30: *Diagrama de Caso de Uso General*

INVENTARIO:

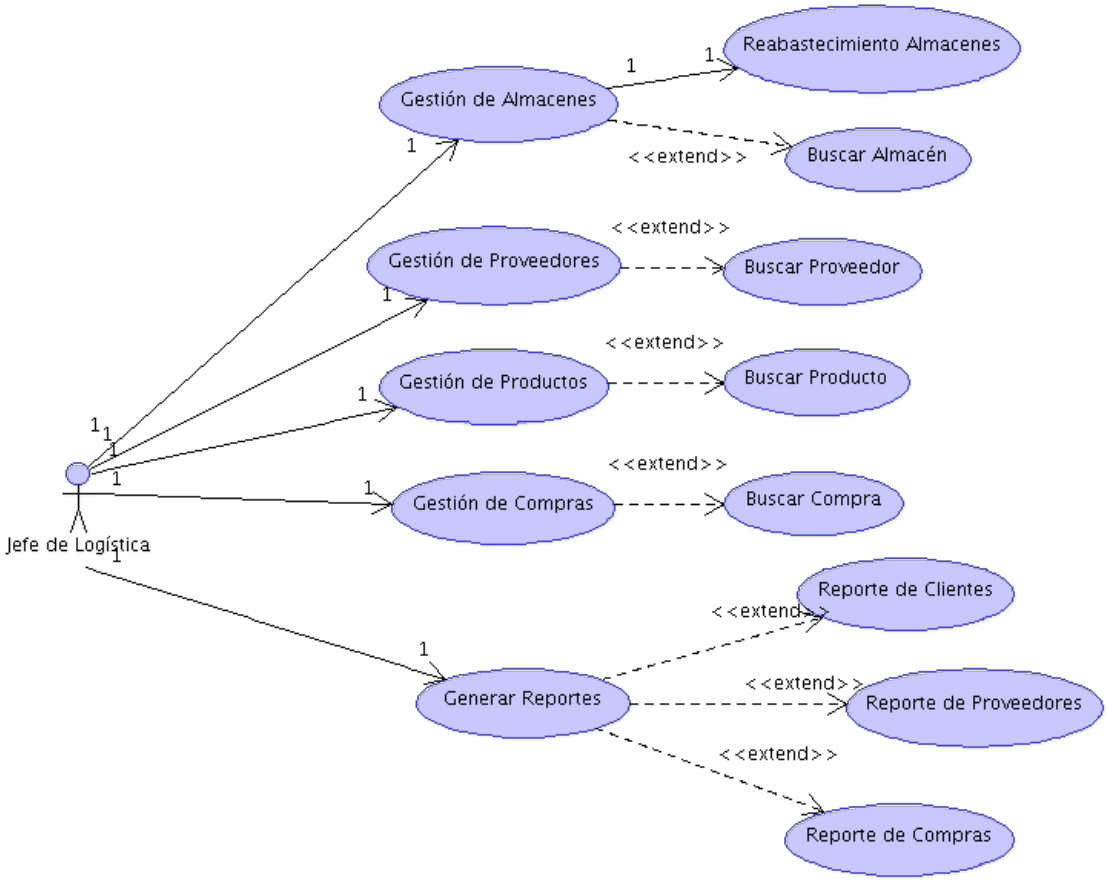


Ilustración 31: *Diagrama de Caso de Uso del Módulo de Inventario*

SEGURIDAD:

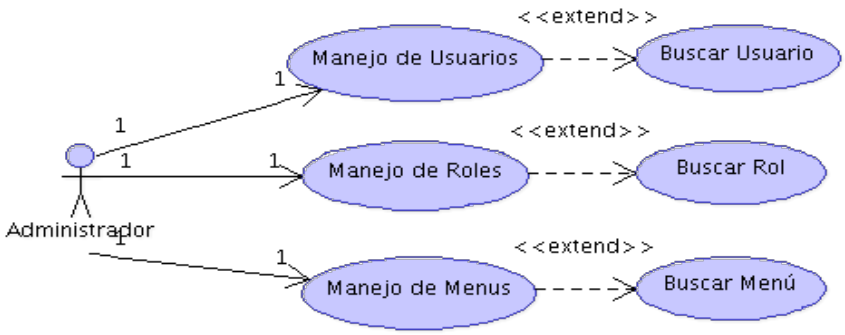


Ilustración 32: *Diagrama de Caso de Uso del Módulo de Seguridad*

4.4 REQUISITOS

A continuación se muestran los artefactos utilizados para declarar los requisitos de software, es decir, el documento visión, el documento glosario y las especificaciones de los casos de uso.

4.4.1 VISIÓN

4.4.1.1 INTRODUCCIÓN

El presente documento es uno de los artefactos que se encuentran dentro de la metodología RUP, se centra en la funcionalidad requerida por los participantes en el proyecto y los usuarios finales.

4.4.1.1.1 PROPÓSITO

El propósito de este documento es recoger, analizar y definir las necesidades de alto nivel y las características del *"Sistema de Planificación de Recursos Empresariales" – FinanSoft*.

Los detalles de cómo el sistema cubre los requerimientos se puede observar en la especificación de los casos de uso y otros documentos adicionales.

4.4.1.1.2 ALCANCE

El documento de Visión se ocupa, como ya se ha descrito, del *"Sistema de Planificación de Recursos Empresariales"* – *FinanSoft*. Dicho sistema será desarrollado por Alcides Rivera Posso.

El sistema permitirá a los encargados de la empresa controlar todo lo relativo a la actividad laboral de la empresa (gestión de productos, administración de clientes, gestión de recursos humanos, etc).

4.4.1.2 POSICIONAMIENTO

4.4.1.2.1 OPORTUNIDAD DE NEGOCIO

Este sistema permitirá a la empresa informatizar el control de sus actividades (compras, ventas, etc.), lo cual supondrá un acceso rápido y sencillo a los datos, gracias a interfaces gráficas sencillas y amigables. Además los datos accedidos estarán siempre actualizados, lo cual es un factor muy importante para poder llevar un control centralizado de la información.

4.4.1.2.2 SENTENCIA QUE DEFINE EL PROBLEMA

El problema de	Controlar el stock existente de los diferentes productos que ofrece la empresa. Gestionar las compras de los clientes. Gestionar los pedidos realizados a los proveedores. Gestionar la facturación de la empresa.

Afecta a	Contador, Bodegueros, Vendedores, Departamento de Recursos Humanos.
El impacto asociado es	Almacenar toda la información referente a las compras y ventas que realiza la empresa, y que esta información este al instante y actualizada en lugares físicamente muy distantes es un proceso prácticamente imposible de realizar en el caso que no este informatizado.
Una solución adecuada sería	Informatizar todo el proceso, usando una red local con una base de datos accesible desde los distintos nodos de la red y generar interfaces amigables y sencillas con las que acceder a dicha base de datos.
Para	Contador, Bodegueros, Vendedores, Departamento de Recursos Humanos.
Quienes	Controlan los procesos de la empresa.
El nombre del producto	Es una herramienta de software.
Que	Almacena la información necesaria para gestionar una empresa de distribución.
No como	El sistema actual.
Nuestro producto	Permite gestionar las distintas actividades de la empresa mediante una interfaz gráfica sencilla y amigable. Además, proporciona un acceso rápido y actualizado a la información desde cualquier punto que tenga acceso a la base de datos.

Tabla 20: *Sentencia que define el problema del Proyecto*

4.4.1.3 DESCRIPCIÓN DE STAKEHOLDERS (PARTICIPANTES EN EL PROYECTO) Y USUARIOS

Para proveer de forma efectiva productos y servicios que se ajusten a las necesidades de los usuarios, es necesario identificar e involucrar a todos los participantes en el proyecto como parte del modelado de requerimientos.

También es necesario identificar a los usuarios del sistema y asegurarse de que el conjunto de participantes en el proyecto los representa adecuadamente.

Esta sección muestra un perfil de los participantes y de los usuarios involucrados en el proyecto, así como los problemas más importantes que estos perciben para enfocar la solución propuesta hacia ellos. No describe sus requisitos específicos ya que éstos se capturan mediante otro artefacto. En lugar de esto proporciona la justificación de por que éstos requisitos son necesarios.

4.4.1.3.1 RESUMEN DE LOS STAKEHOLDERS

Nombre	Descripción	Responsabilidades
Gerente de la empresa	Representante global de la empresa.	El Stakeholder realiza: • Representa a todos los usuarios posibles del sistema. • Seguimiento del desarrollo del proyecto. • Aprueba requisitos y funcionalidades.

Tabla 21: *Resumen de los Stakeholders del Proyecto*

4.4.1.3.2 RESUMEN DE LOS USUARIOS

Nombre	Descripción	Stakeholder
ACT1 Administrador	Responsable de la gestión de usuarios, roles y la administración del menú del sistema.	STK1 Seguridad

ACT2 Contador	Encargado de llevar la contabilidad de la empresa.	STK2 Contabilidad
ACT3 Vendedor	Encargado de la facturación y gestión de clientes.	STK3 Facturación
ACT4 Jefe de Almacén	Supervisor del buen funcionamiento del almacén y de gestionar las incidencias de los pedidos, ya sea tratando con otro almacén, o bien en contacto con el Jefe de Logística.	STK4 Inventario
ACT5 Jefe de Logística	Responsable del Departamento de Logística, encargado de la gestión del almacén central, del aprovisionamiento del resto de almacenes y del contacto con los proveedores.	STK4 Inventario
ACT6 Jefe de Recursos Humanos	Responsable de la gestión de empleados.	STK5 Recursos Humanos

Tabla 22: *Resumen de los Usuarios del Proyecto*

4.4.1.3.3 ENTORNOS DE USUARIOS

Los usuarios necesitan disponer de un navegador web únicamente. Los reportes pueden ser generados tanto en pdf, html, procesadores de texto y hojas de cálculo.

4.4.1.3.4 PERFIL DE LOS STAKEHOLDERS

REPRESENTANTE DEL ÁREA TÉCNICA Y SISTEMAS DE INFORMACIÓN

Representante	Gerente de la empresa.
Descripción	Representante global de la empresa.
Tipo	Experto en negocios.
Responsabilidades	Encargado de mostrar las necesidades de cada usuario del sistema. Además, lleva a cabo un seguimiento del desarrollo del proyecto y aprobación de los requisitos y funcionalidades del sistema.
Criterio de éxito	A definir por el cliente.
Grado de participación	Revisión de requerimientos, estructura del sistema.

Tabla 23: *Representante del Área Técnica del Proyecto*

4.4.1.3.5 PERFILES DE USUARIO

ADMINISTRADOR

Representante	STK1 SEGURIDAD
Descripción	Administrador.
Tipo	Experto en sistemas.
Responsabilidades	Responsable de la gestión de usuarios, roles y la administración del menú del sistema.
Criterio de éxito	A definir por el cliente.
Grado de participación	A definir por el cliente.

Tabla 24: *Usuario Administrador del Proyecto*

CONTADOR

Representante	STK2 CONTABILIDAD
Descripción	Empleado del Departamento de Contabilidad.
Tipo	Usuario experto.
Responsabilidades	Encargado de llevar la contabilidad de la empresa.
Criterio de éxito	A definir por el cliente.
Grado de participación	A definir por el cliente.

Tabla 25: *Usuario Contador del Proyecto*

VENDEDOR

Representante	STK3 FACTURACIÓN
Descripción	Vendedor.
Tipo	Usuario experto.
Responsabilidades	Encargado de la facturación y gestión de clientes.
Criterio de éxito	A definir por el cliente.
Grado de participación	A definir por el cliente.

Tabla 26: *Usuario Vendedor del Proyecto*

JEFE DE ALMACÉN

Representante	STK2 INVENTARIO
Descripción	Jefe de almacén.
Tipo	Usuario experto.
Responsabilidades	Encargado de llevar la contabilidad de la empresa. Supervisar el buen funcionamiento del almacén y de gestionar las incidencias de los pedidos, ya sea tratando con otro almacén, o bien en contacto con el Jefe de Logística. Capacidad de toma de decisiones en cuanto a distribución de mercaderías desde otro almacén y cancelación de pedidos que han sido atendidos.

Criterio de éxito	A definir por el cliente.
Grado de participación	A definir por el cliente.

Tabla 27: *Usuario Jefe de almacén del Proyecto*

JEFE DE LOGÍSTICA

Representante	STK4 INVENTARIO
Descripción	Jefe de Logística.
Tipo	Usuario experto.
Responsabilidades	Responsable del Departamento de Logística, encargado de la gestión del almacén central, del aprovisionamiento del resto de almacenes y del contacto con proveedores. Control de estadísticas para la optimización de recursos.
Criterio de éxito	A definir por el cliente.
Grado de participación	A definir por el cliente.

Tabla 28: *Usuario Jefe de Logística del Proyecto*

JEFE DE RECURSOS HUMANOS

Representante	STK5 RECURSOS HUMANOS
Descripción	Jefe de Recursos Humanos.
Tipo	Usuario experto.
Responsabilidades	Responsable de la gestión de empleados.
Criterio de éxito	A definir por el cliente.
Grado de participación	A definir por el cliente.

Tabla 29: *Usuario Jefe de Recursos Humanos del Proyecto*

4.4.1.4 DESCRIPCIÓN GLOBAL DEL PRODUCTO

4.4.1.4.1 PERSPECTIVA DEL PRODUCTO

El producto a desarrollar es un *"Sistema de Planificación de Recursos Empresariales"* – *FinanSoft* para la empresa, con la intención de agilizar su funcionamiento. Las áreas a tratar por el sistema son: contabilidad, inventario, facturación, recursos humanos y seguridad.

4.4.1.4.2 RESUMEN DE LAS CARACTERÍSTICAS

A continuación se mostrará un listado de los beneficios que obtendrá el cliente a partir del producto.

Beneficio del cliente	**Características que lo apoyan**
Gestión automatizada del stock del almacén.	Sistema de optimización del stock en el almacén y previsión de pedidos.
Mayor agilidad en los pedidos dando la posibilidad de hacerlo vía web.	Aplicación web desde la cual poder realizar los pedidos.
Mayor facilidad para la gestión de recursos humanos.	Base de datos centralizada con la información de todo el personal.
Mayor control de los productos.	El sistema gestionará los productos del almacén, generará reportes.
Seguridad.	El ingreso del sistema se controla por medio de un usuario y contraseña, se controla el acceso a las opciones a través de permisos de roles, y demás seguridades que proveen los marcos de trabajo.

Tabla 30: *Resumen de las características del Proyecto*

4.4.1.4.3 SUPOSICIONES Y DEPENDENCIAS

El *"Sistema de Planificación de Recursos Empresariales"* – *FinanSoft* va a integrarse a un sistema ya existente en las instalaciones de la empresa llamado Sinafin, el cual esta conectado a una base de datos en Microsft FoxPro 6.0.

Para garantizar el término de una transacción de reserva se pretende minimizar el peso y uso de gráficos que impidan el uso ágil de la página. Todo término de transacción satisfactorio o insatisfactorio será notificado inmediatamente al usuario.

4.4.1.5 DESCRIPCIÓN GLOBAL DEL PRODUCTO

CSW1: CONTABILIDAD

El departamento de contabilidad tendrá acceso a todo el módulo de *Contabilidad.*

CSW2: FACTURACIÓN

El departamento de ventas tendrá acceso al módulo de *Facturación* que se encarga de realizar las ventas y manejar los datos de los clientes.

CSW3: INVENTARIO

El departamento de logística dirige y gestiona el almacén centralizado de la empresa, que es el abastecimiento principal del resto de almacenes. Este departamento dispondrá del módulo de *Inventario* que automatizará el proceso de reposición de stocks de los almacenes y el reabastecimiento de los distintos almacenes, tanto el central como las sucursales mediante los proveedores de la empresa.

CSW4: RECURSOS HUMANOS

El departamento de recursos humanos es encargado de la gestión de personal. Los empleados con rol de recursos humanos tendrán acceso al módulo de *Recursos Humanos* en el que se darán de alta, de baja y se modificarán datos de la plantilla.

CSW5: SEGURIDAD

El administrador del sistema será encargado del manejo del módulo de *Seguridad* y de la administración de los servidores donde se encontrará el sistema.

4.4.1.6 RESTRICCIONES

Las restricciones que el sistema presenta y advierte a sus usuarios es la necesidad de contar una adecuada conectividad entre las sucursales y un navegador web (firefox, opera, safari, internet explorer, etc).

El hardware requerido por las diferentes sucursales es el adecuado y además se ha efectuado una inversión de un servidor para el alojamiento del servidor de aplicaciones y del sistema de gestión de base de datos relacional.

4.4.1.7 OTROS REQUISITOS DEL PROYECTO

4.4.1.7.1 ESTÁNDARES APLICABLES

Lenguaje para el diseño de páginas WEB: HTML avanzado (compatible desde cualquier navegador web).

Sistema de Gestión de Base de Datos Relacional: PostgreSQL.
Protocolo de Comunicación: TCP/IP versión 4.

4.4.1.7.2 REQUISITOS DE SISTEMA

Como ya se menciono anteriormente el requerimiento esencial para los usuarios es contar con una conexión adecuada de internet o un canal de datos, y un navegador web (firefox, opera, safari, internet explorer, etc).

4.4.1.7.3 REQUISITOS DE DESEMPEÑO

El requerimiento mayor de rendimiento estará dado por la facilidad de acceso al sistema de la empresa. Se ha considerado un diseño vistoso pero ligero de peso.

4.4.1.7.4 REQUISITOS DE ENTORNO

Los dispositivos de red y servidores, tendrán que ser fijados en un rack para cumplir con algunos estándares de cableado estructurado. Para los productos se necesita un ambiente con temperaturas: *-5<15<+5 °C*.

4.4.1.8 REQUISITOS DE DOCUMENTACIÓN

4.4.1.8.1 MANUAL DE USUARIO

Los manuales de usuario podrán ser descargados directamente desde el sistema. El manual de usuario contendrá toda la documentación de la instalación, uso y mantenimiento del sistema. Será enfocado a cada rol de usuario, ya que cada uno maneja diferentes tipos de información.

4.4.1.8.2 AYUDA EN LÍNEA

La ayuda en línea podrá ser accedida de dos formas:

- Usando los diferentes hipervínculos situados cerca de las opciones relevantes del sistema.

- Mediante el acceso a nuestra página web, con la posibilidad de consultas y sugerencias.

4.4.1.8.3 GUÍAS DE INSTALACIÓN, CONFIGURACIÓN Y FICHERO LÉAME

Se encuentra en los anexos digitales del proyecto.

4.4.2 GLOSARIO

4.4.2.1 INTRODUCCIÓN

Este documento recoge todos y cada uno de los términos manejados a lo largo de todo el proyecto de desarrollo del *"Sistema de Planificación de Recursos Empresariales"* – *FinanSoft*. Se trata de un diccionario informal de datos y definiciones de la nomenclatura que se maneja, de tal modo que se crea un estándar para todo el proyecto.

4.4.2.1.1 PROPÓSITO

El propósito de este glosario es definir con exactitud y sin ambigüedad la terminología manejada en el proyecto de desarrollo del *"Sistema de Planificación de Recursos Empresariales"* – *FinanSoft*. También sirve como guía de consulta para la clarificación de los puntos conflictivos o poco esclarecedores del proyecto.

4.4.2.1.2 ALCANCE

El alcance del presente glosario se extiende a todos los módulos definidos en el *"Sistema de Planificación de Recursos Empresariales"* – *FinanSoft*. De tal modo que la terminología empleada en el departamento de contabilidad, recursos humanos, ventas, etc; se refleja con claridad aquí.

4.4.2.2 ORGANIZACIÓN DEL GLOSARIO

El presente glosario está organizado por definiciones de términos ordenados de forma ascendente según la ordenación alfabética tradicional.

4.4.2.3 DEFINICIONES

A continuación se presentan todos los términos manejados a lo largo de todo el proyecto de desarrollo de un *"Sistema de Planificación de Recursos Empresariales" – FinanSoft*.

CLIENTE EXTERNO:
El cliente externo es el cliente propiamente dicho, es decir, la visión que ofrece RUP del modelo de caso de uso del negocio, el cliente externo representa uno de los tantos agentes externos con los que interactúan la empresa. Por tanto, el cliente externo es el comprador de los artículos, que puede ser cualquier almacén.

ERP (ENTERPRISE RESOURCE PLANNING):
Los sistemas de planificación de recursos empresariales, son sistemas de información gerenciales que integran y manejan muchos de los negociados asociados con las operaciones de producción y de los aspectos de distribución de una compañía comprometida en producción de bienes o servicios.

IDE (INTEGRATE DEVELOPMENT ENVIROMENT):
Entorno de desarrollo o diseño integrado o entorno de depuración integrada; proporciona las instalaciones para el desarrollo de software: Editor de código fuente, compilador, herramientas de automatización y un editor.

LÓGICA DEL NEGOCIO:

Mucha información necesita ser tratada por una lógica específica para poder convertirse en información útil para el usuario.

ORM (OBJECT RELATION MAPPING):

Es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional. Crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Posibilita (básicamente herencia y polimorfismo).

POJO (PLAIN OLD JAVA OBJECT):

Enfatiza el uso de clases simples en el marco de una revalorización de la programación.

TRANSFORMACIÓN DE DATOS:

Permite operaciones sofisticadas en agrupación de cantidades, porcentajes de los totales generales y más. Datos ordenados, filtrados, según la necesidad del usuario.

4.4.3 ESPECIFICACIÓN DEL CASO DE USO: "GESTIONAR PRODUCTO"

Este caso de uso permite al jefe del almacén tener una gestión de productos para luego poder administrarlos en el funcionamiento de nuestro sistema, el jefe de almacén podrá crear, editar y eliminar os productos según sea el criterio de la empresa.

4.4.3.1 FLUJO DE EVENTOS

4.4.3.1.1 FLUJOS BÁSICOS

El sistema muestra la interfaz *"Gestión de Productos"* con los criterios de búsqueda que son los campos: código, nombre, genérico, tipo; además de las opciones buscar, editar, nuevo, eliminar.

CREAR PRODUCTO

1. El caso de uso comienza cuando el jefe de almacén solicita *"Crear"* en la interfaz *"Gestión de Productos"*.
2. El sistema muestra la interfaz *"Crear Producto"*.
3. El jefe de almacén ingresa todos los datos del producto.
4. El jefe de almacén elige la opción *"Guardar"* para guardar el nuevo producto creado.
5. El sistema guarda registro nuevo del producto ingresado.
6. El sistema genera mensaje de confirmación de creación de producto.
7. El sistema regresa a la interfaz *"Gestión de Productos"*.

EDITAR PRODUCTO

1. El jefe de almacén ingresa el criterio de búsqueda del producto en la interfaz *"Gestión de Productos"*.
2. El jefe de almacén selecciona el botón *"Buscar"*.
3. Se ejecuta el caso de uso extendido *"Buscar Producto"*.
4. El sistema muestra los productos y sus datos en una tabla en la interfaz

"Gestión de Productos".

5. El jefe de almacén presiona *"Editar"* en el producto que desea modificar.
6. El jefe de almacén modifica los datos del producto según criterio.
7. El jefe de almacén selecciona *"Guardar"*.
8. El sistema guarda los datos modificados y generá un mensaje *"Actualización Completa"*.
9. El jefe de almacén presiona *"Aceptar"*.
10. El sistema regresa a la interfaz *"Gestión de Productos"*.

ELIMINAR PRODUCTO

1. El jefe de almacén ingresa al criterio de búsqueda del producto en la interfaz *"Gestión de Productos"*.
2. El jefe de almacén selecciona buscar.
3. Se ejecuta el caso de uso extendido *"Buscar Producto"*.
4. El sistema muestra los productos y sus datos en una tabla en la interfaz *"Gestión de Productos"*.
5. El jefe de almacén selecciona el producto que desea eliminar.
6. El jefe de almacén presiona el botón *"Eliminar"*.
7. El sistema muestra mensaje *"Seguro que desea eliminar el producto seleccionado"*.
8. El jefe de almacén confirma presionando el botón *"Aceptar"*.
9. El sistema elimina el producto seleccionado de la base de datos.

4.4.3.1.2 FLUJOS ALTERNATIVOS

ERROR FALTA DE INGRESAR DATOS OBLIGATORIOS

En el punto cuatro de crear producto, cuando el usuario selecciona *"Guardar*" sin haber llenado todos los campos requeridos, el sistema muestra un mensaje de error *"Campo requerido"* en cada uno de los campos que son necesarios.

NO COLOCA CRITERIO DE BÚSQUEDA

Si en editar producto y eliminar producto el jefe de almacén presiona buscar sin ingresar un criterio de búsqueda, el sistema mostrará en la tabla todos los productos ingresados hasta el momento.

4.4.3.2 PRECONDICIONES

El jefe de almacén tiene que estar logeado en el sistema.
Que existan productos en la base de datos.

4.4.3.3 POSTCONDICIONES

El sistema ha actualizado la lista de productos.

4.4.3.4 PUNTOS DE EXTENSIÓN

El sistema llama al caso de uso *"Buscar Producto"*.

4.4.4 ESPECIFICACIÓN DEL CASO DE USO: "LOGIN – LOGOUT"

Este caso de uso permite al usuario ingresar al sistema para poder obtener los privilegios necesarios para manejar el sistema y cerrar su sesión concluida sus tareas.

4.4.4.1 FLUJO DE EVENTOS

4.4.4.1.1 FLUJO BÁSICOS

LOGIN

1. El sistema solicita la cuenta de usuario.
2. El usuario ingresa su cuenta de usuario en la interfaz *"Autentificación de Usuarios"*.
3. EL sistema solicita contraseña de usuario.
4. El usuario ingresa su contraseña de usuario en la interfaz *"Autentificación de Usuarios"*.
5. El usuario selecciona la opción *"Aceptar"* y el caso de uso finaliza.

LOGOUT

1. El usuario selecciona la opción *"Cerrar Sesión"*.
2. El sistema solicita la confirmación del *"Cierre de Sesión"*.
3. Se cierra la sesión y se muestra la interfaz *"Autentificación de Usuarios"*.

4.4.4.1.2 FLUJOS ALTERNATIVOS

ERROR DE CUENTA DE USUARIO

Del punto dos en el caso de que ingrese de manera incorrecta su cuenta de usuario el sistema muestra un mensaje de error.

ERROR DE CONTRASEÑA

Del punto cuatro en el caso de que ingrese de manera incorrecta su contraseña el sistema muestra un mensaje de error.

SUBFLUJO: ERROR DE INGRESO DE CONTRASEÑA POR TERCERA VEZ

En el caso de que el usuario ingrese su contraseña por tercera vez en forma incorrecta, el sistema restringirá el ingreso de esa cuenta y se llamará al administrador del sistema.

4.4.4.2 PRECONDICIONES

El usuario debe tener una cuenta de usuario.

4.4.4.3 POSTCONDICIONES

El sistema queda conectando mediante una sesión y se tiene acceso a las opciones que tiene con sus privilegios.

4.4.5 REQUERIMIENTOS

4.4.5.1 STAKEHOLDERS

Los representantes de los usuarios y portavoces de las necesidades de la empresa son los stakeholders. En este proyecto solamente se ha tratado con un stakeholder como representante de los usuarios y necesidades de la empresa, sin embargo se han dividido representativamente.

La matriz de atributos de los stakeholders es la siguiente:

Requerimientos	Representante	Ubicación
STK1: Contabilidad	Departamento de Contabilidad	Documento Visión
STK2: Facturación	Departamento de Ventas	Documento Visión
STK3: Inventario	Departamento de Logística	Documento Visión
STK4: Recursos Humanos	Departamento de Recursos Humanos	Documento Visión
STK5: Seguridad	Administrador	Documento Visión

Tabla 31: *Matriz de atributos de los Stakeholders del Proyecto*

4.4.5.2 ACTORES

Se define este requerimiento para listar los usuarios potenciales del sistema, en este proyecto se han definido los siguientes actores: *Contador*, *Jefe de Almacén, Vendedor, Jefe de Logística, Jefe de Recursos Humanos y Administrador.*

Requerimientos	Ubicación	Módulo
ACT1: Contador	Documento Visión	Contabilidad
ACT2: Vendedor	Documento Visión	Facturación
ACT3: Jefe de Almacén	Documento Visión	Inventario
ACT4: Jefe de Logística	Documento Visión	Inventario
ACT5: Jefe de Recursos Humanos	Documento Visión	Recursos Humanos
ACT6: Administrador	Documento Visión	Seguridad

Tabla 32: *Matriz de atributos de los Actores del Proyecto*

4.4.5.3 CARACTERÍSTICAS DE SOFTWARE

Las características de software son las necesidades de los usuarios propuestas por los stakeholders de la empresa, son los requisitos que debe cumplir el sistema para satisfacer las necesidades de los trabajadores y de la empresa.

Las características definidas son las que aparecen en la matriz de atributos, siendo las indicadas como subcaracterísticas las derivadas según una clasificación jerárquica.

Requerimientos	Asignado a
CSW1: Contabilidad Contabilidad	
CSW1.1: Gestión del plan de cuentas Gestión del plan de cuentas	Equipo completo de análisis, desarrollo e implementación
CSW1.2: Gestión del libro diario Gestión del libro diario	Equipo completo de análisis, desarrollo e implementación
CSW2: Facturación Facturación	
CSW2.1: Gestión de datos de clientes Gestión de los datos de los clientes	Equipo completo de análisis, desarrollo e implementación
CSW2.2: Gestión de ventas Gestión de ventas	Equipo completo de análisis, desarrollo e implementación
CSW3: Inventario Inventario	
CSW3.1: Gestión de datos de proveedores Gestión de los datos de los proveedores	Equipo completo de análisis, desarrollo e implementación
CSW3.2: Gestión de datos de productos Gestión de los datos de los productos	Equipo completo de análisis, desarrollo e implementación
CSW3.3: Gestión de compras Gestión de compras	Equipo completo de análisis, desarrollo e implementación
CSW4: Recursos Humanos Recursos Humanos	
CSW4.1: Gestión de personal Gestión de personal	Equipo completo de análisis, desarrollo e implementación
CSW5: Seguridad Seguridad	
CSW5.1: Manejo de usuarios Manejo de usuarios	Equipo completo de análisis, desarrollo e implementación
CSW5.2: Manejo de roles Manejo de roles	Equipo completo de análisis, desarrollo e implementación
CSW5.3: Manejo de menús Manejo de menús	Equipo completo de análisis, desarrollo e implementación

Tabla 33: *Matriz de atributos de las caraterísticas del Proyecto*

4.4.5.4 CASOS DE USO

Derivados de las características de software, son el resultado del análisis de las necesidades de los usuarios. La matriz de atributos es la siguiente:

Requerimientos	**Asignado a**
ECU1: Gestión del Plan de Cuentas.	Equipo de desarrollo.
ECU2: Buscar Cuenta	Equipo de desarrollo.
ECU3: Gestión del Libro Diario.	Equipo de desarrollo.
ECU4: Buscar Asiento.	Equipo de desarrollo.
ECU5: Gestión del Cliente.	Equipo de desarrollo.
ECU6: Buscar Cliente.	Equipo de desarrollo.
ECU7: Gestión de Ventas.	Equipo de desarrollo.
ECU8: Buscar Venta.	Equipo de desarrollo.
ECU9: Gestión de Proveedores.	Equipo de desarrollo.
ECU10: Buscar Proveedor.	Equipo de desarrollo.
ECU11: Gestión de Productos.	Equipo de desarrollo.
ECU12: Buscar Producto.	Equipo de desarrollo.
ECU13: Gestión de Compras.	Equipo de desarrollo.
ECU14: Buscar Compras.	Equipo de desarrollo.
ECU15: Gestión de Personal.	Equipo de desarrollo.
ECU16: Buscar Empleado.	Equipo de desarrollo.
ECU17: Manejo de Usuarios.	Equipo de desarrollo.
ECU18: Buscar Usuario.	Equipo de desarrollo.
ECU19: Manejo de Roles.	Equipo de desarrollo.
ECU20: Buscar Rol.	Equipo de desarrollo.
ECU21: Manejo de Menús.	Equipo de desarrollo.
ECU22: Buscar Menú.	Equipo de desarrollo.
ECU23: Login – Logout.	Equipo de desarrollo.
ECU24: Generar Reportes.	Equipo de desarrollo.

Tabla 34: *Matriz de atributos de los Casos de Uso del Proyecto*

4.4.5.5 CLASES

Requerimientos	Ubicación
CLS1: Asiento	Base de Datos
CLS2: Cliente	Base de Datos
CLS3: ClienteReferencia	Base de Datos
CLS4: Cuenta	Base de Datos
CLS5: DetalleAsiento	Base de Datos
CLS6: DetalleFactura	Base de Datos
CLS7: Empleado	Base de Datos
CLS8: Localidad	Base de Datos
CLS9: Mayor	Base de Datos
CLS10: Menu	Base de Datos
CLS11: Producto	Base de Datos
CLS12: RetencionFuente	Base de Datos
CLS13: RetencionIva	Base de Datos
CLS14: RetencionTransporte	Base de Datos
CLS15: Tipo	Base de Datos
CLS16: TipoLocalidad	Base de Datos
CLS17: Usuario	Base de Datos

Tabla 35: *Matriz de atributos de las Clases del Proyecto*

4.5 ANÁLISIS/DISEÑO

A continuación se presentan los modelos definidos en RUP como modelo de datos y modelo de análisis/diseño (Diagrama de Clases, Modelo Relacional).

4.5.1 DIAGRAMA DE CLASES

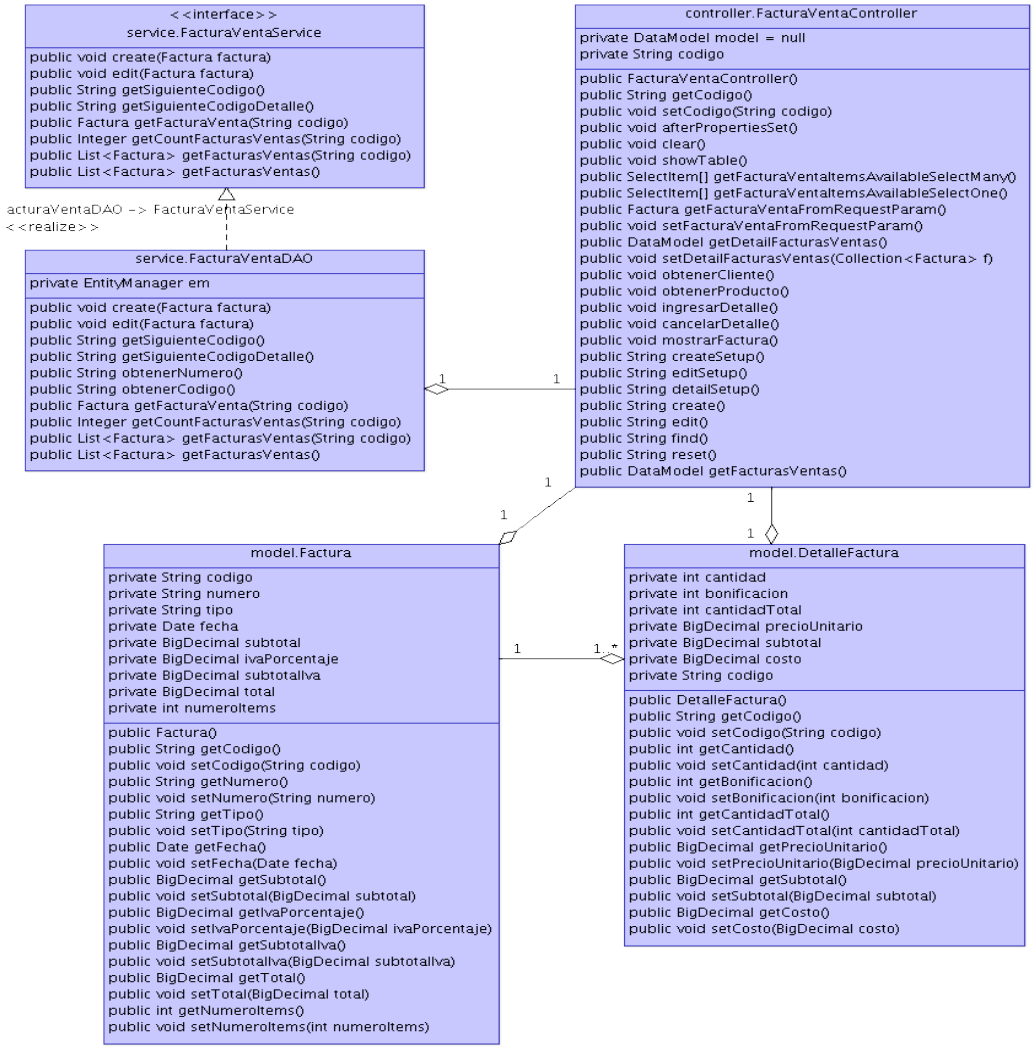


Ilustración 33: *Diagrama de Clase: Punto de Venta del Proyecto*

4.5.2 MODELO ENTIDAD RELACIÓN

CONTABILIDAD:

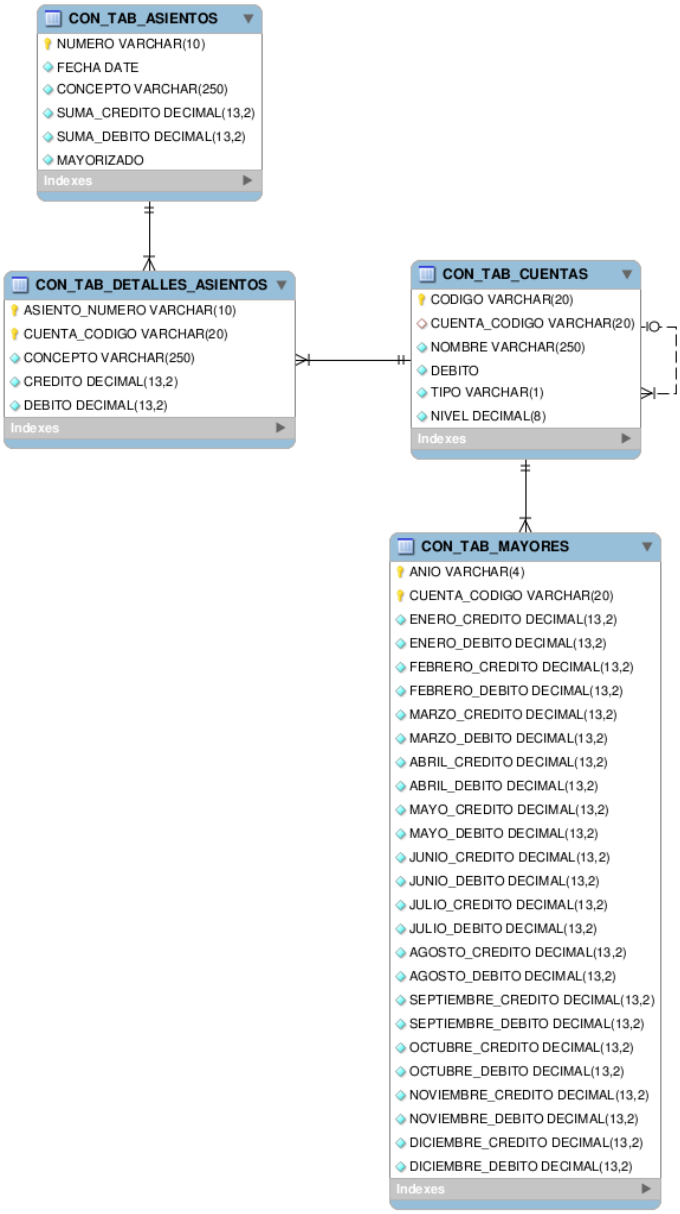


Ilustración 34: *Diagrama Relacional del Módulo de Contabilidad del Proyecto*

GENERAL

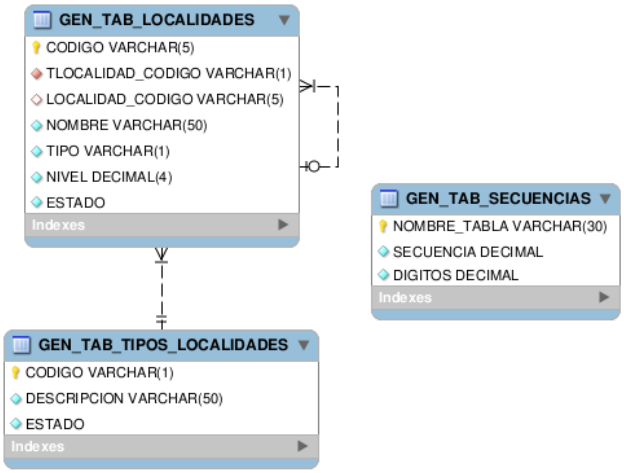


Ilustración 35: *Diagrama Relacional del Módulo General del Proyecto*

RECURSOS HUMANOS

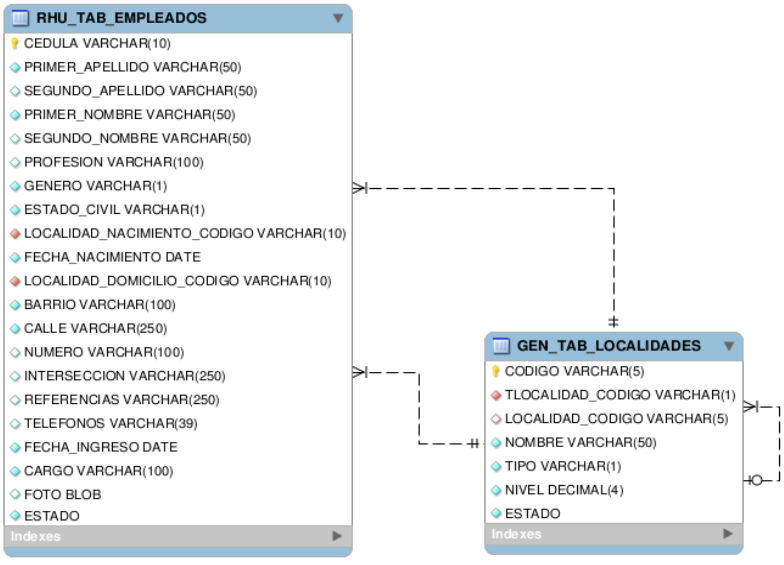


Ilustración 36: *Diagrama Relacional del Módulo de Recursos Humanos del Proyecto*

INVENTARIO

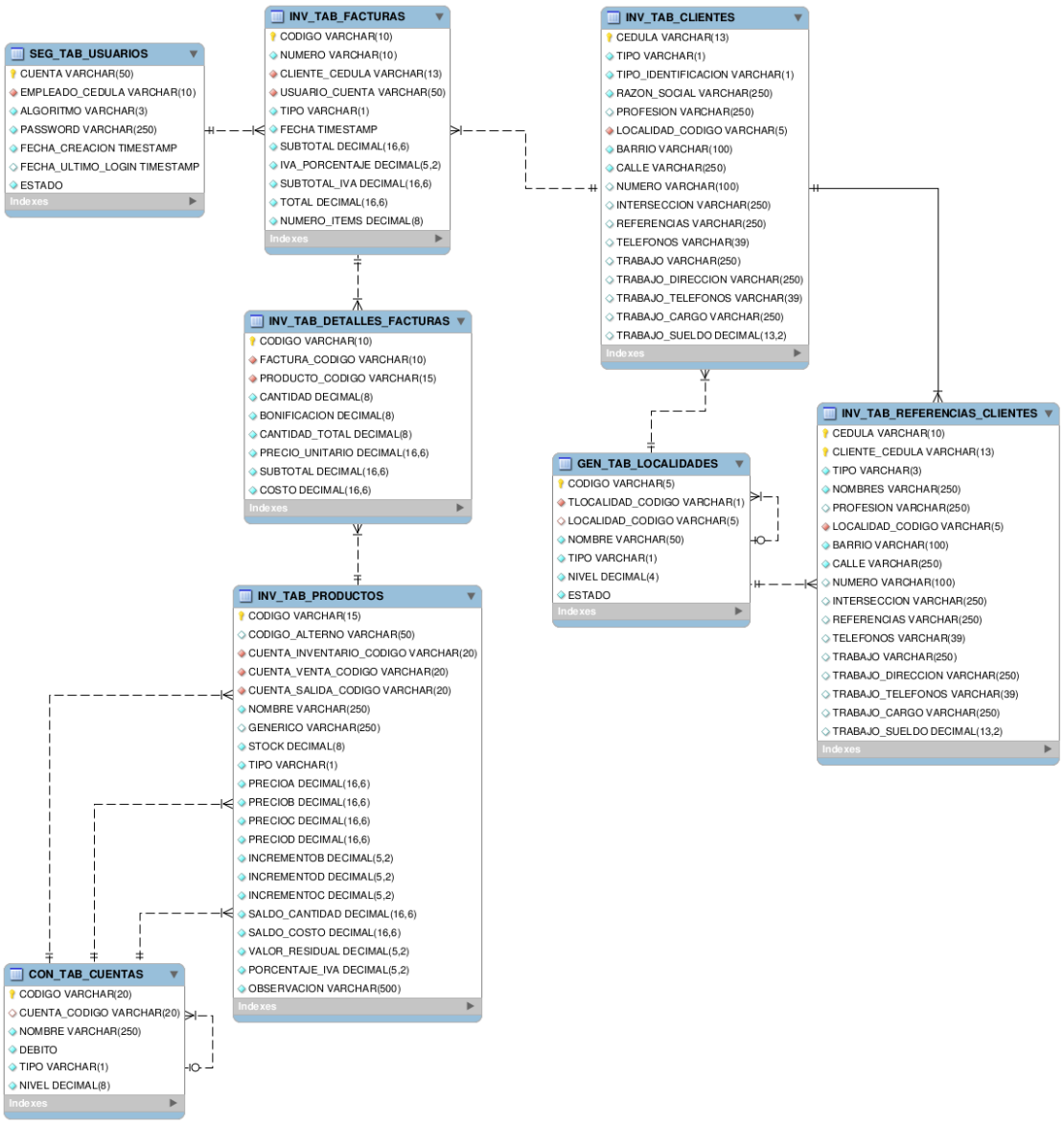


Ilustración 37: *Diagrama Relacional del Módulo de Inventario del Proyecto*

SEGURIDAD

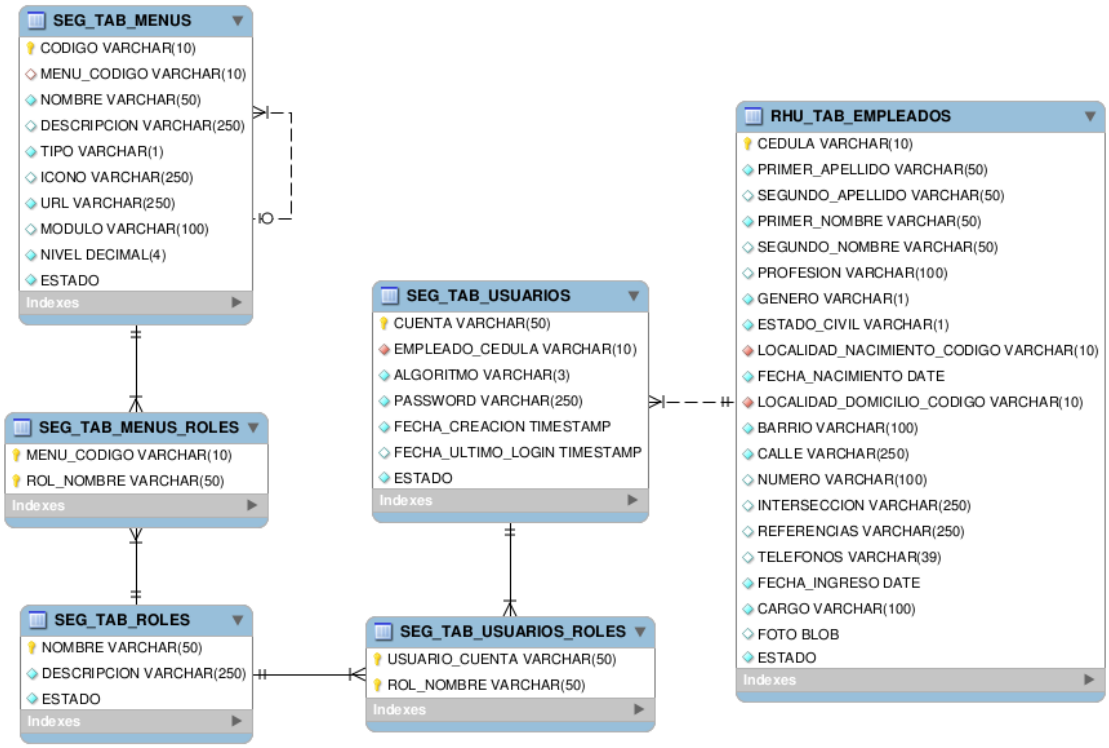


Ilustración 38: *Diagrama Relacional del Módulo de Seguridad del Proyecto*

4.6 IMPLEMENTACIÓN

A continuación se presentan en los modelos definidos en RUP como prototipos de interfaces de usuario, diagrama de componentes y diagrama de despliegue del proyecto.

4.6.1 PROTOTIPOS DE INTERFACES DE USUARIO

A continuación se presentan los prototipos de interfaces gráficas de usuario diseñadas para la aplicación final.

PRINCIPAL

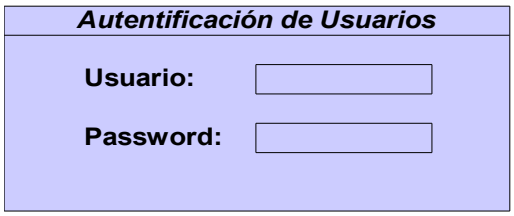


Ilustración 39: *Interfaz de Usuario: Login del Proyecto*

MENÚ PRINCIPAL

Ilustración 40: *Interfaz de Usuario: Menú Principal del Proyecto*

LISTADOS

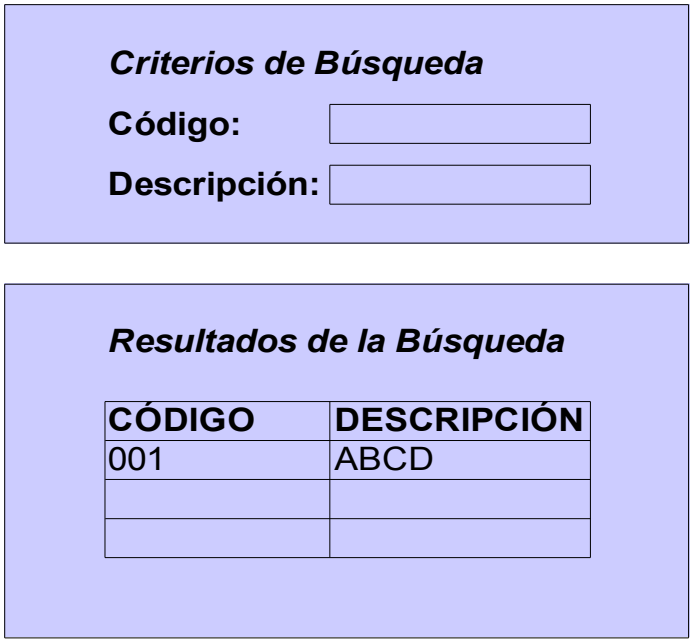


Ilustración 41: *Interfaz de Usuario: Listado de Items del Proyecto*

CREACIÓN Y EDICIÓN

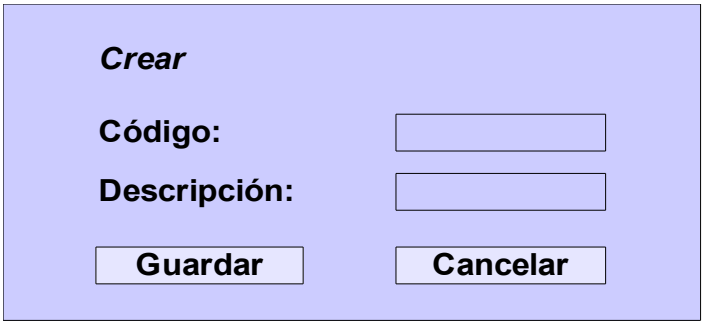


Ilustración 42: *Interfaz de Usuario: Creación y Edición de Items del Proyecto*

4.6.2 DIAGRAMA DE COMPONENTES

Se muestra la disposición de las partes integrantes de la aplicación y las dependencias entre los distintos módulos de la aplicación.

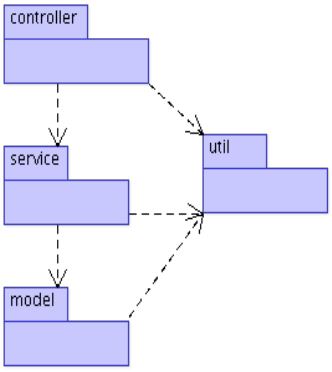


Ilustración 43: *Diagrama de Componentes del Proyecto*

4.6.3 DIAGRAMA DE DESPLIEGUE

Se muestra la disposición de la arquitectura de la aplicación.

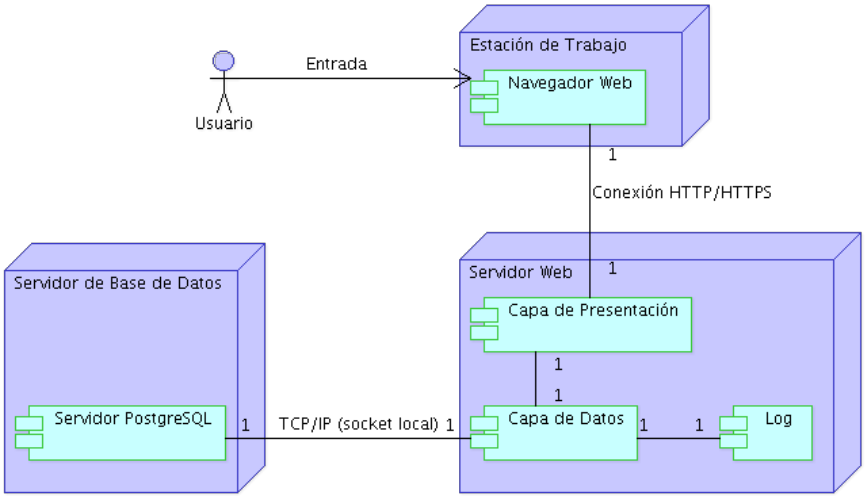


Ilustración 44: *Diagrama de Despliegue del Proyecto*

4.6.4 ARQUITECTURA DE SOFTWARE

4.6.4.1 INTRODUCCIÓN

4.6.4.1.1 PROPÓSITO

Este documento tiene como propósito describir la vista general de la arquitectura de software del *Sistema de Planificación de Recursos Empresariales – FinanSoft* para lo cual se utilizan vistas arquitectónicas para representar los diferentes aspectos del sistema.

4.6.4.1.2 ALCANCE

Este documento de arquitectura de software proporciona una descripción del "Sistema de Planificación de Recursos Empresariales" - FinanSoft. Este documento ha sido generado directamente del análisis de los casos de uso a automatizar y del modelo de diseño.

4.6.4.2 REPRESENTACIÓN ARQUITECTÓNICA

Modelo de Caso de Uso: Describe los procesos que brindarán al negocio la funcionalidad automatizada deseada y cómo funcionan internamente, contiene el modelo de caso de uso.

Modelo de Análisis: Describe un primer bosquejo de las clases de análisis que servirán de soporte para el diseño.

Modelo de la Experiencia del Usuario: Describe las pantallas del sistema, el contenido dinámico de pantallas y como el usuario navega a través de las pantallas para ejecutar las funcionalidades del sistema.

Modelo de Diseño: Describe las partes arquitectónicas significativas del modelo de diseño, tales como su descomposición en módulos y paquetes.

En el desarrollo del sistema se ha utilizado el patrón de arquitectura Modelo Vista Controlador (MVC).

4.6.4.3 OBJETIVOS ARQUITECTÓNICOS Y COACCIONES

Los principales objetivos del *"Sistema de Planificación de Recursos Empresariales"* – *FinanSoft* es la automatización de todos sus procesos para un mejor control de todos los productos que cuenta el almacén, además se podrá obtener reportes actualizados de todos los procesos.

Todos los requerimientos estipulados en el documento de la Visión deben ser tomados en consideración durante el desarrollo de la arquitectura.

La construcción principal del diseño y de la implementación ha sido que la aplicación debe funcionar bajo una plataforma que consiste en los componentes siguientes:

1. **Lenguaje de programación:** Java.
2. **Entorno de Desarrollo:** NetBeans 6.9
3. **Servidor:** SuSE Linux Enterprise Server.
4. **Sistema de Gestión de Base de Datos:** Postgresql 8.4.3

5. **Servidor de Aplicaciones:** GlassFish 3.1
6. **Arquitectura Tecnológica:** JEE (Java Edición Empresarial) 6.

4.6.4.4 VISTA DE CASOS DE USO

Nos presenta una vista de los casos de uso de la arquitectura de software. La vista de casos de uso es la entrada importante de la selección de contexto y/o los casos de uso que son el punto de una interacción. Los casos de uso del *"Sistema de Planificación de Recursos Empresariales" – FinanSoft*.

4.6.4.4.1 CASOS DE USO ARQUITECTÓNICAMENTE SIGNIFICATIVOS

Los caso de uso arquitectónicamente significativos son aquellos que representan las partes más críticas de la arquitectura del sistema y demuestran la funcionalidad del sistema. Además de lo ya mencionado, para el *"Sistema de Planificación de Recursos Empresariales" – FinanSoft* los casos de uso significativos han sido priorizados en base al soporte que brindan a las metas del negocio.

REGISTRAR INGRESO PRODUCTO NUEVO

Permitirá al almacén el ingreso al sistema de todos los productos que adquieren y así poder tener actualizado su stock.

REGISTRAR SALIDA

Permitirá al almacén tener registrado las salidas de productos por ventas. Teniendo un mayor control de su stock.

4.6.4.5 VISTA LÓGICA

Esta sección describe la estructura lógica del sistema. Empieza con la descripción de la arquitectura y después presenta sus elementos estructurales y del comportamiento dominantes.

4.6.4.5.1 ELEMENTOS DEL MODELO ARQUITECTURALMENTE SIGNIFICATIVO

El “Sistema de Planificación de Recursos Empresariales” – FinanSoft se ha descompuesto en los siguientes módulos: Contabilidad, Facturación, Inventario, Recursos Humanos y Seguridad.

Cada uno de los componentes del negocio se divide más a fondo en las tres capas del patrón de arquitectura Modelo Vista Controlador (MVC).

1. Lógica de la presentación,
2. Lógica del negocio, y
3. Lógica de la integración.

Es decir la arquitectura descompone los sistemas a la largo de dos dimensiones:

1. La primera dimensión está a lo largo de las líneas de la funcionalidad del sistema.

2. La segunda dimensión está a lo largo de las capas comúnmente reconocidas que separan tres clases de preocupaciones:

a) Preocupaciones de la presentación, o cómo manejar la comunicación con el usuario y controlar su acceso a los servicios y a los recursos de sistema.

b) Preocupaciones de negocio, o cómo organizar los elementos del sistema que realizan funciones de los servicios del negocio y del sistema, y

c) Preocupaciones de la integración, o cómo conectar los elementos del sistema con el mecanismo de la persistencia, otros sistemas, los dispositivos físicos, etc.

4.7 PRUEBAS

A continuación se muestran las especificaciones de casos de prueba funcionales de los casos de uso incluidos en el proyecto de desarrollo de software.

4.7.1 ESPECIFICACIÓN DEL CASO DE PRUEBA: "LOGIN - LOGOUT"

4.7.1.1 INTRODUCCIÓN

En la presente sección se detallan las pruebas a realizarse para los distintos escenarios del caso de uso *"Login – Logout".*

4.7.1.1.1 PROPÓSITO

Probar que el caso de uso *"Login – Logout"* está correctamente implementado y que se cumplen las especificaciones funcionales y no funcionales.

4.7.1.1.2 ALCANCE

Sólo se prueban los escenarios mencionados en el documento de Especificación de Caso de Uso: *"Login – Logout".*

4.7.1.1.3 DEFINICIONES, ACRÓNIMOS Y ABREVIACIONES

Para el presente sección tener en cuenta los siguientes términos:

Autenticar: Se refiere al hecho de haber ingresado sus datos para que la aplicación pueda identificar que la persona que intenta acceder a su contenido es quien dice ser. A esta acción se lo denomina técnicamente Login.

Sesión: Período de tiempo de actividad que un usuario pasa en el sistema, desde que hace el Login hasta que hace Logout.

4.7.1.2 ESCENARIOS DE PRUEBA

El presente documento contiene los distintos escenarios que se detallaron en la especificación del caso de uso *"Login – Logout"* y cada escenario tiene una breve descripción de lo que trata, el flujo de actividades que contiene los pasos a realizarse para cumplir el escenario y los puntos de control que indican los pasos donde evaluar exhaustivamente.

4.7.1.2.1 ESCENARIO: FLUJO BÁSICO

Validar que el usuario pueda iniciar y terminar su sesión de manera correcta.

Descripción: Es el escenario ideal del caso de uso, no deberían presentarse errores.

PRECONDICIONES

El usuario no debe tener su sesión activa en el sistema. Se deben haber creado el usuario y sus datos indicando como dato de entrada.

DATOS DE ENTRADA

Se accederá al sistema con el usuario *"prueba"*[73] cuya contraseña es *"123"*. Su perfil es *Vendedor.*

FLUJO DE ACTIVIDADES

Paso	Instrucción	Resultado esperado
1	El usuario ingresa a la interfaz de inicio de sesión accediendo a la url de la aplicación.	El sistema muestra la interfaz de inicio de sesión o logeo en la que se solicita cuenta de usaurio y contraseña.
2	El usuario ingresa a su cuenta de usuario y contraseña en la interfaz de logeo y selecciona la opción *"Aceptar"*.	El sistema muestra menú de opciones de módulos de la interfaz principal.
3	El usuario selecciona la opción *"Cerrar Sesión"*.	El sistema solicita la confirmación de cierre de sesión.
4	El usuario confirma el cierre de sesión dando click en *"Aceptar"*.	El sistema muestra la interfaz de inicio de sesión.

Tabla 36: *Flujo de actividades del escenario Flujo Básico del Caso de Prueba "Login – Logout" del Proyecto*

73 Este usuario es solo para fines que persigue este documento. Luego de realizadas las pruebas, el usuario debe ser borrado de la base de datos.

PUNTOS DE REVISIÓN

Paso	Punto de Control	Validación a realizar
1	Carga de la página de autentificación (login).	Al momento de intentar acceder al sistema, el sistema revisa si la sesión del usuario existe u si está activa. Como no lo está, muestra la página de Login con el campo para el ingreso del usuario y otro para la contraseña.
2	Acceso del usuario al sistema.	El usuario ha iniciado sesión en el sistema, con lo que habrá generado una cookie con la data referente a la sesión.
3	El sistema solicita confirmación de cierre de sesión.	Verificar que las opciones de confirmación estén correctamente habilitadas.
4	El sistema muestra la interfaz de inicio de sesión.	El sistema debe mostrar la interfaz de logeo.

Tabla 37: *Punto de revisión del escenario Flujo Básico del Caso de Prueba "Login – Logout" del Proyecto*

4.7.1.2.2 ESCENARIO: ERROR DE CUENTA DE USUARIO

Verificar el reconocimiento de las cuentas de usuario validando su existencia en la base de datos.

Descripción: Comprobar que una cuenta se encuentra o no registrada en la base de datos.

PRECONDICIONES

La cuenta de usuario no ha sido registrada en la base de datos.

DATOS DE ENTRADA

Un usuario *"xyz"* con clave *"123"*.

FLUJO DE ACTIVIDADES

Paso	Instrucción	Resultado esperado
1	El usuario ingresa su cuenta de usuario.	El sistema muestra mensaje de error *"Cuenta incorrecta"* y retorna a la interfaz que lo solicito.

Tabla 38: *Flujo de actividades del escenario Error de Usuario del Caso de Prueba "Login – Logout" del Proyecto*

PUNTO DE CONTROLADOR

Al final de dicha acción muestra el mensaje correspondiente.

PUNTOS DE REVISIÓN

Paso	Punto de Control	Validación a realizar
1	Muestra: mensaje de error.	Verificar que los datos sean los del set de datos comprobar que el sistema muestre el mensaje de acuerdo al error ocurrido.

Tabla 39: *Puntos de revisión del escenario Error de Usuario del Caso de Prueba "Login – Logout" del Proyecto*

4.7.1.2.3 ESCENARIO: ERROR DE CONTRASEÑA

Verificar el reconocimiento de las contraseñas de usuarios validando su existencia en la base de datos.

Descripción: Comprobar que una cuenta se encuentra o no registrada en la base de datos.

PRECONDICIONES

El usuario a emplearse tiene una contraseña.

DATOS DE ENTRADA

Se intentará acceder al sistema con el usuario *"prueba"* cuya contraseña es *"123"*, sin embargo se utilizará la contraseña *"987"*.

FLUJO DE ACTIVIDADES

Paso	Instrucción	Resultado esperado
1	El usuario ingresa su contraseña.	El sistema muestra mensaje de error *"Contraseña incorrecta"* y retorna a la interfaz que lo solicito.

Tabla 40: *Flujo de actividades del escenario Error de Contraseña del Caso de Prueba "Login – Logout" del Proyecto*

PUNTOS DE CONTROL

Al final de dicha acción muestra el mensaje correspondiente.

PUNTOS DE REVISIÓN

Paso	Punto de Control	Validación a realizar
1	Muestra: mensaje de error.	Verificar que los datos sean los del set de datos comprobar que el sistema muestre el mensaje de acuerdo al error ocurrido.

Tabla 41: *Puntos de revisión del escenario Error de Contraseña del Caso de Prueba "Login – Logout" del Proyecto*

4.7.1.2.4 ESCENARIO: ERROR DE CONTRASEÑA INGRESADA POR TERCERA VEZ

PRECONDICIONES

Se ha intentado el ingreso dos veces, con el resultado fallido.

DATOS DE ENTRADA

Se intentará acceder al sistema con el usuario *"prueba"* cuya contraseña es *"123"*, sin embargo se utilizará la contraseña *"987"* tres veces consecutivas.

FLUJO DE ACTIVIDADES

Paso	Instrucción	Resultado esperado
1	El usuario ingresa su contraseña por tercera vez.	Se muestra el mensaje *"Usuario Bloqueado"* y se alerta al administrador.

Tabla 42: *Flujo de actividades del escenario Error de Contraseña tercera vez del Caso de Prueba "Login – Logout" del Proyecto*

PUNTOS DE REVISIÓN

Paso	Punto de Control	Validación a realizar
1	El sistema restringe el ingreso del usuario.	Verificar que la restricción sea efectiva y que este habilitada correctamente la alerta al administrador.

Tabla 43: *Puntos de revisión del escenario Error de Contraseña tercera vez del Caso de Prueba "Login – Logout" del Proyecto*

4.7.1.2.5 ESCENARIO: CAMPOS VACÍOS

DATOS DE ENTRADA

En principio no se usa usuario, luego no se usa contraseña y finalmente no se usan ambos.

FLUJO DE ACTIVIDADES

Paso	Instrucción	Resultado esperado
1	Se ingresa la cuenta de usuario, pero no la contraseña.	
2	Se confirma inicio de sesión.	Indica que falta ingresar la contraseña.
3	Se ingresa únicamente la contraseña.	
4	Se confirma inicio de sesión.	Indica que falta ingresar la cuenta de usuario.
5	No se ingresa ni cuenta de usuario, ni contraseña.	
6	Se confirma inicio de sesión.	Indica que falta ingresar la cuenta de usuario y la contraseña.

Tabla 44: *Flujo de actividades del escenario Campos vacíos del Caso de Prueba*

PUNTOS DE REVISIÓN

Paso	Punto de Control	Validación a realizar
2,4,6	Acceso denegado.	No se permite ingresar el usuario al sistema.

Tabla 45: *Puntos de revisión del escenario Campos vacíos del Caso de Prueba "Login – Logout" del Proyecto*

CAPÍTULO V

5 CONCLUSIONES Y RECOMENDACIONES

5.1 VERIFICACIÓN DE HIPÓTESIS

HIPÓTESIS:

La utilización de una metodología la Arquitectura de Software en el desarrollo de aplicaciones, permitirá mejorar la calidad del software final, en: eficiencia, confiabilidad, usabilidad, mantenibilidad, expandibilidad, interoperabilidad, reusabilidad, integridad y portabilidad.

Por el estudio realizado durante la elaboración de este trabajo de grado podemos decir que la hipótesis ha sido comprobada, ya que al utilizar la metodología de Arquitectura de Software en el desarrollo de nuestra aplicación se obtuvo un software de calidad en los siguientes parámetros:

Eficiencia: El sistema funciona con mucha rapidez en los procesos.

Confiabilidad: Los resultados que nos proporciona el sistema tiene un alto nivel de confiabilidad.

Usabilidad: El sistema es bastante fácil de utilizar ya que posee interfaces muy prácticas.

Mantenibilidad y Expandibilidad: El sistema al ser elaborado siguiendo un patrón de arquitectura de software, su mantenimiento por parte de otros desarrolladores es muy sencillo.

Interoperatibilidad: Esta condición ha sido puesta en práctica ya que se trabaja simultaneamente con otro sistema que posee la empresa.

Reusabilidad: El sistema proporciona código fuente que podrá ser reutilizado en otros sistemas que la empresa desarrolle en un futuro.

Integridad: La seguridad ha sido uno de los factores más tomados en cuenta en el desarrollo de nuestro aplicativo por lo tanto podemos afirmar que nuestro sistema es integro.

Portabilidad: El sistema al haber sido desarrollado en una arquitectura tecnológica multiplataforma nuestro aplicativo podrá ser ejecutado en cualquier plataforma tecnológica.

5.2 CONCLUSIONES

1. La Arquitectura de Software se basa en el tratamiento de los estilos, el desarrollo de los lenguajes de descripción arquitectónica, la formulación de metodologías y los patrones de diseño.

2. La Arquitectura de Software proporciona un modelo relativamente pequeño, fácil de entender, de la forma en que un sistema se estructura y sus componentes se entienden entre sí; se puede lograr el acoplamiento de este modelo en sistemas de requerimientos similares y promover una reutilización a gran escala.

3. El arquitecto de software tiene entre sus tareas fundamentales, el diseño o selección de la arquitectura, su representación, evaluación y análisis.

4. De los Lenguajes de Descripción Arquitectónica actuales ninguno se impuso ni en la academia ni el mercado, por lo que existe un compás de espera con: UML 2.0, Lenguajes Específicos de Dominio, XML y Web Semántica; ya que el proceso de producción de ADL's se ha desacelerado.

5. Las descripciones arquitectónicas aportan nuevas oportunidades para el análisis, incluyendo verificaciones de consistencia del sistema, conformidad con las restricciones impuestas por un estilo, conformidad con atributos de calidad, análisis de dependencias y análisis específicos de dominio y negocio.

6. Cada uno de los patrones de arquitectura de software están destinados a un campo específico del desarrollo de aplicaciones de software por lo tanto no podemos decir cual de ellos es el mejor.

7. La decisión de escoger a Java EE (Java Enterprise Edition) como arquitectura tecnológica en el desarrollo de nuestro aplicativo,es debido a ser la mejor que actualmente nos proporciona el mercado.

8. La hipótesis planteada en mi trabajo de grado fue comprobada ya que se logró un software de altísima calidad, que cumplió con las expectativas esperadas.

9. El contar con este aplicativo realizado en aplicación web nos permite acceder a la información extrayendo reportes y estadísticas en línea de manera rápida, atendiendo a solicitudes realizadas por la empresa.

10. Implementar el software permitió contar con un sistema de calidad para el desempeño y productividad de sus empleados, y se simplifico el control de acceso a la información financiera, contando con información organizada, centralizada y de fácil acceso.

11. Este proyecto permitió un cambio organizacional previo y durante la implementación, atendiendo los requerimientos en los servicios de contabilidad, facturación e inventario de productos.

5.3 RECOMENDACIONES

1. Proponer la creación de la materia de Arquitectura de Software dentro de la malla curricular de la carrera de Ingeniería en Sistemas Computacionales, en los últimos años de la misma para así poder tener una idea clara de como administrar proyectos informáticos.

2. Si un proyecto no ha logrado una arquitectura del sistema, incluyendo su justificación, el proyecto no debe empezar ha gran escala. Si se especifica la arquitectura como un elemento ha entregar, se la puede usar a lo largo del proceso de desarrollo y del proceso de mantenimiento.

3. Plantear al Centro de Capacitación Continua de la Facultad de Ingeniería en Ciencias Aplicadas la apertura de cursos orientados a tecnologías de Software Libre.

4. Promover el desarrollo de software utilizando la Arquitectura de Software, en los proyectos de titulación por parte de los docentes de la facultad.

5. El arquitecto debe hacer hincapié y centrarse en los atributos de calidad del sistema, los llamados requisitos no funcionales. Debe verificar las necesidades de los stakeholders con respecto al rendimiento, escalabilidad, mantenimiento, seguridad, internacionalización entre otros.

6. El arquitecto debe explicar los motivos detrás de la elección de determinados elementos arquitectónicos en detrimento de los demás. También debe realizar el análisis “buy vs. Build”, es decir, comprar o utilizar un sistema en lugar de construirlo desde cero.

7. En el caso de trabajar en sistemas y aplicaciones empresariales y corporativas se recomienda tener al menos un “documento de arquitectura de software”. Dicho documento debe contener solamente la información esencial, tal como: elementos y los principales componentes del sistema, capas del sistema, mecanismos arquitecturales necesarios para el sistema, tecnologías elegidas para hacer frente a cada capa y el mecanismo y la motivación detrás de estas opciones, componentes comprados o de código abierto que se utilizarán y la forma en que se tomo la decisión, descripción de los principales patrones de diseño utilizados y por qué la elección, definición de la forma en que se acceso a sistemas externos y/o ligados.

8. Siempre se debe recordar que las actividades relacionadas a la arquitectura de software se hacen de manera iterativa e incremental. Por lo tanto, se debe tener cuidado de no caer en el modelo de cascada y estar durante largos períodos de tiempo tan sólo escribiendo el documento de la arquitectura.

9. Recomendar a la Universidad el cambio del sistema informático integrado actual a una plataforma de software libre lo cual permitirá entre otras cosas un mejor manejo del desarrollo de software en equipo, podrá disponer del sistema en el internet, utilizar una arquitectura de software y acatará la sugerencia de su utilización dictada por parte del estado.

5.4 POSIBLES TEMAS DE TESIS

- Evaluación de la arquitectura de software adoptada por un determinado aplicativo.

- Estudio del Modelo Vista Controlador.

- Estudio comparativo de las Vistas Arquitectónicas.

- Estudio de la implementación de la arquitectura de software en Data WareHouse.

- Estudio comparativo de los diferentes repositorios de desarrollo de software.

REFERENCIAS BIBLIOGRÁFICAS

[LIB01]: Jan Bosch, Design and use of Software Architecture, Addison - Wesley, 2000
[LIB02]: Paul Clements, "A Survey of Architecture Description Languages". Procedings of the International Workshop of Software Specification and Design, Alemanio, 1996
[LIB03]: Steve Vestal, A cursory overview and comparison of four Architecture Languages, Honeywell Technology Center, 1993
[LIB04]: John A. Zachman, A Framework for Information Systems Architecture, IBM Systems Journal, 1987
[LIB05]: Philippe Kruchten, The 4+1 View Model of Architecture, IEEE Software, 1995
[LIB06]: Grady Booch, James Rumbaugh e Ivar Jacobson, El Lenguaje Unificado de Modelado, Addison-Wesley, 1999
[LIB07]: Len Bass, Paul Clements y Rick Kazman, Software Architecture in Practice. Reading, Addison-Wesley, 1998
[LIB08]: Mary Shaw y Paul Clements, "A field guide to Boxology: Preliminary classification of architectural styles forsoftware systems, , 1996
[LIB09]: Mark Klein y Rick Kazman, Attribute-based architectural styles, Carnegie Mellon University, 1999
[LIB10]: Roger Pressman, Ingeniería del Software: Un enfoque práctico, McGraw Hill, 2001
[LIB11]: Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King y Shlomo Angel, A Pattern Language: Towns/Building/Construction, Oxford University Press, 1977
[LIB12]: Ramirez, A, Introducción a los Patrones de Diseño, Creative Commons, 2004
[LIB13]: Buschmann, F. Meunier, R. Rohnert, H. Sommerlad, P. Stal, M, Pattern-Oriented Software Architecture. A System of Patterns, Wiley, 1996
[WWW01]: http://www.janbosch.com/Jan%20Bosch.html
[WWW02]: http://www.sei.cmu.edu/staff/clements
[WWW03]: http://es.wikipedia.org/wiki/IEEE
[WWW04]: http://en.wikipedia.org/wiki/Philippe_Kruchten
[WWW05]: http://en.wikipedia.org/wiki/Jim_Lane_(Microsoft)
[WWW06]: http://www.gestiopolis.com/.../arquitectura-de-software-como-disciplina-cientifica.htm
[WWW07]: http://www.infobiz.com.ar/arquitectura_software_argentina.html
[WWW09]: http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=7
[WWW10]: http://es.wikipedia.org/wiki/Stakeholders
[WWW11]: http://www.gestiopolis.com/recursos/experto/catsexp/pagans/mar/no%2013/lookandfeel.htm
[WWW12]: http://www.desarrolloweb.com/articulos/1622.php
[WWW13]: http://www.ibm.com/developerworks/library/w-berry

[WWW14]: http://www.willydev.net/descargas/prev/ADL.pdf
[WWW15]: http://www.ldc.usb.ve/~abianc/materias/ci4712/Software%20Architecture-kazman.pdf
[WWW16]: http://triana.escet.urjc.es/aspf/Tema4-1.pdf
[WWW17]: http://download.microsoft.com/download/c/2/c/c2ce8a3a-b4df-4a12-ba18-7e050aef3364/070717-Real_World_SOA.pdf
[WWW18]: http://dialnet.unirioja.es/servlet/fichero_tesis?codigo=18529&orden=0
[WWW19]: http://www.monografias.com/trabajos27/evolucion-patrones/evolucion-patrones.shtml
[WWW20]: http://mario-chavez.blogspot.com/2008/07/la-arquitectura-de-cebolla-parte-1.html
[WWW21]: http://www.ldc.usb.ve/~teruel/ci4712/clases2000/clase6.html
[WWW22]: http://www.dossier-andreas.net/software_architecture/pipe_and_filter.html
[WWW23]: http://es.wikipedia.org/wiki/Ken_Thompson
[WWW24]: http://es.wikipedia.org/wiki/Douglas_McIlroy
[WWW25]: , http://www.vico.org/pages/PatronsDisseny/Pattern%20Pipes%20and%20Filters/, ,
[WWW26]: http://www.dossier-andreas.net/software_architecture/pipe_and_filter.html
[WWW27]: http://www.dossier-andreas.net/software_architecture/blackboard.html
[WWW28]: http://www.vico.org/pages/PatronsDisseny/Pattern%20Blackboard/
[WWW29]: http://www.dcc.uchile.cl/~mmarin/revista-sccc/sccc-web/Vol5/wis1.pdf
[WWW30]: http://www.vico.org/pages/PatronsDisseny/Pattern%20Broker/
[WWW31]: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.9852&rep=rep1&type=pdf
[WWW32]: http://apuntes.rincondelvago.com/trabajos_global/informatica/37/
[WWW33]: http://www.acm.org/crossroads/espanol/xrds12-4/arqcentric.html
[WWW34]: http://es.wikipedia.org/wiki/Mapeo_objeto_relacional
[WWW35]: http://www.tecnoretales.com/programacion/que-es-doctrine-orm/
[WWW36]: http://www.ufg.edu.sv/ufg/theorethikos/Julio04/mdp6.html
[WWW37]: http://www.proactiva-calidad.com/java/patrones/mvc.html
[WWW38]: http://es.wikipedia.org/wiki/Swing_%28biblioteca_gr%C3%A1fica%29
[WWW39]: http://es.wikipedia.org/wiki/Microsoft_Foundation_Classes
[WWW40]: http://es.wikipedia.org/wiki/Modelo_Vista_Controlador
[WWW41]: www.glosarium.com/term/705,14,xhtml
[WWW42]: http://es.wikipedia.org/wiki/Callback_%28inform%C3%A1tica%29
[WWW43]: http://es.wikipedia.org/wiki/Framework
[WWW44]: http://es.wikipedia.org/wiki/Licencia_MIT
[WWW45]: http://es.wikipedia.org/wiki/Apache_License
[WWW46]: http://www.gnu.org/licenses.es.html
[WWW47]: http://eprints.rclis.org/archive/00013953/01/Evaluaci%C3%B3n_de_software_libre_para_la_gesti%C3%B3n_de_bibliograf%C3%ADa_-_completo.pdf
[WWW48]: http://es.wikipedia.org/wiki/Mozilla_Public_License

[WWW49]: http://maestriainfopn.unlugar.com/inginfo/arquitectura-tecnologica.ppt
[WWW50]: http://es.wikipedia.org/wiki/Java_EE
[WWW51]: http://es.wikipedia.org/wiki/Sun_Microsystems
[WWW52]: http://es.wikipedia.org/wiki/Java_Community_Process
[WWW53]: http://es.wikipedia.org/wiki/Software_development_kit
[WWW54]: http://java.sun.com/javaee/
[WWW55]: http://es.wikipedia.org/wiki/Microsoft_.NET