UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

ESCUELA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES

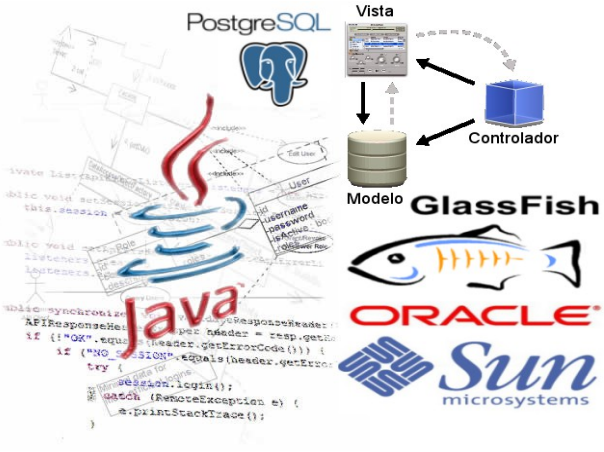
INFORME TÉCNICO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS COMPUTACIONALES

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

DICIEMBRE DE 2010

RESUMEN

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.

Arquitectura de Software es 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.

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: Capas (Layered Systems), Tuberías y Filtros (Pipe and Filter), Pizarra (Blackboard) Broker (Arquitectura Orientada a Servicios) y Modelo Vista Controlador (Model View Controller).

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. Entre las Arquitecturas Tecnológicas más utilizadas tenemos JEE (Java Enterprise Edition), Microsoft .NET y PHP.

Utilizando la metodología de Arquitectura de Software en el desarrollo de nuestras aplicaciones se logrará un software de calidad con los siguientes parámetros: eficiencia, confiabilidad, usabilidad, mantenibilidad, expandibilidad, interoperabilidad, reusabilidad, integridad y portabilidad.

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.

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.

ÍNDICE DE CONTENIDOS

1 ARQUITECTURA DE SOFTWARE

La Arquitectura de Software es 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.

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.

1.1 EL ARQUITECTO DE SOFTWARE

El término Arquitecto de Software se ha convertido en el título de moda en toda empresa de sistemas o con un 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.

1.2 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.

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.

1.3 LENGUAJES DE DESCRIPCIÓN ARQUITECTÓNICA

Un Lenguaje de Descripción Arquitectónica (ADL) 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. Un ADL es un lenguaje que proporciona características para modelar la arquitectura conceptual de un sistema software, distintiva de la implementación del sistema.

Criterios de Definición de un ADL: Los criterios que vamos a tomar en cuenta son los que se encuentran en Vestal, quien sostiene que un ADL debe modelar o soportar los siguientes conceptos: componentes, conexiones, composición jerárquica, paradigmas de computación, paradigmas de comunicación, modelos formales y subadyacentes, soporte de herramientas; y composición automática de código aplicado.

Lenguajes: Ninguno de los lenguajes 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.

1.4 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.

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.

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: vista lógica, vista de proceso, vista física y vista 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.

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. La vista arquitectónica propuesta por Grady Booch, James Rumbaugh e Ivar Jacobson en su introducción a UML en donde se encuentra: vista de casos de uso, vista de diseño, vista de interacción, vista de implementación y vista de despliegue.

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: estructura de módulo, estructura lógica o conceptual, estructura física, estructura de uso, estructura de llamados, flujos de datos, flujo de control y estructura de clases.

1.5 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."

Entre los principales estilos arquitectónicos tenemos:

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, entre los repositorios de estos tenemos: repositorio pasivo y repositorio activo (pizarra). 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*.

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. 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.*

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, dentro de este estilo tenemos dos subestilos como son: programa principal/subprograma, llamada de procedimiento remoto.

Arquitectura 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.

Arquitectura 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. 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.

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. 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.

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.

2 PATRONES DE ARQUITECTURA DE SOFTWARE

2.1 DEFINICIONES

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.

2.2 ARQUITECTURA EN CAPAS

"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."

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.

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 ARQUITECTURA TUBOS Y FILTROS

"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".

Estructura: El patrón de arquitectura Tubos y Filtros esta conformado por:

- **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.

- **Filtro:** Es el encargado de filtrar o transformar los datos que recibe a través de las tuberías

- **Bomba (Entrada):** También llamado productor es el origen de datos. Puede ser archivos, bases de datos, o dispositivos de entrada.

- **Sumidero (Salida):** Se conoce también como consumidor es el objetivo de los datos, puede ser a archivos, bases de datos o dispositivos de salida.

2.4 ARQUITECTURA EN PIZARRA

"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."

Estructura: Un sistema de pizarra tiene frecuentemente la siguiente estructura:

- **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.

- **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.

- **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.

2.5 ARQUITECTURA DE BROKER

"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."

Estructura: El patrón Broker comprende de seis componentes participantes:

- **Cliente:** 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*.

- **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.

- **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.

- **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

- **Proxy 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.

- **Proxy 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.

2.6 ARQUITECTURA MODELO VISTA CONTROLADOR

"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."

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

- **Modelo:** Es el objeto que representa los datos del programa. 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.

- **Vista:** Es el objeto que maneja la presentación visual (Interfaz Gráfica de Usuario) 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.

- **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.

3 ARQUITECTURAS TECNOLÓGICAS

3.1 JEE (JAVA ENTERPRISE EDITION)

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."*

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. Los EJB's son unas de las API's 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. Hay tres tipos de EJB's:

- **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 y persistencia gestionada por el bean.

- **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) y sin sesión (stateless).

- **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.

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.

- **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.

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"* 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.

3.2 .NET (MICROSOFT)

"*.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."*

Los principales componentes de Microsoft .NET son:

- **Entorno Común de Ejecución (CRL):** .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.

- **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.

3.3 PHP

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

La actual versión de PHP 5 incluye las siguientes mejoras: mejor soporte para programación rientada a objetos con PDO, mejoras en el rendimiento, mejor soporte para MySql con extensión completamente reescrita, mejor soporte a XML (Xpath, DOM, etc), soporte nativo para SQLite, soporte integrado para SOAP, iteradores de datos, manejo de excepciones; y mejoras con la implementación de Oracle.

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"*.

Frameworks: Los frameworks más importantes que posee PHP son:

- **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.

4 APLICATIVO

4.1 GESTIÓN DEL PROYECTO

El proyecto debe proporcionar una respuesta para el diseño del prototipo de un *"Sistema de Planificación de Recursos Empresariales"* – *FinanSoft.* El cual consta de los siguientes módulos:

Contabilidad: Plan de Cuentas, Jornalización, Mayorización, Balances y Estados Financieros.
Facturación: Clientes y Ventas.
Inventario: Productos, Proveedores y Compras.
Recursos Humanos: Inventario de Personal.
Seguridad: Usuarios, Roles y Menús.

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.

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.

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.

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.

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.

Características de software: Las características de software son las propuestas por el stakeholder de la empresa son los siguientes:

- **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. Dispondrá del módulo de *Inventario* que automatizará el proceso de reposición de stocks de los almacenes y el reabastecimiento de los distintos almacenes.

- **CSW4: RECURSOS HUMANOS:** El departamento de recursos humanos es encargado de la gestión de personal.

- **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.

Stackholders: 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.

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.*

Objetivos: 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. 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.

Cada uno de los componentes del negocio se divide más a fondo en las tres capas del patrón de arquitectura Modelo Vista Controlador (MVC): lógica de la presentación, lógica del negocio, y lógica de la integración.

Los estilos arquitectónicos que fueron utilizados para el desarrollo del prototipo son: Arquitectura Orientada a Objetos y Arquitectura Orientada a Aspectos.