Universidad de las Ciencias Informáticas
Facultad 6

Título: Sistema Informático de los Tribunales Militares de Región: Diseño Arquitectónico.

Trabajo de Diploma para Optar por el Título de

Ingeniero en Ciencias Informáticas

Autores:

Yiselis Díaz Pérez
Yaiser Enrique Betancourt Aldana

Tutores:

Ing. Elián Cutiño Díaz

M.Sc. David Ardite Martínez

Co-tutor:

Ing. José Antonio Castaño Guevara

Ciudad de la Habana
Junio del 2007“
Año 49 de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaramos que somos las únicas autoras de este trabajo y autorizamos a la dirección de los Tribunales Militares y al grupo UCI-MINFAR de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio

Para que así conste firmo la presente a los ____ días del mes de Junio del año 2007.

Autores:
Yiselis Díaz Pérez

_____

Yaiser Enrique Betancourt Aldana

_____

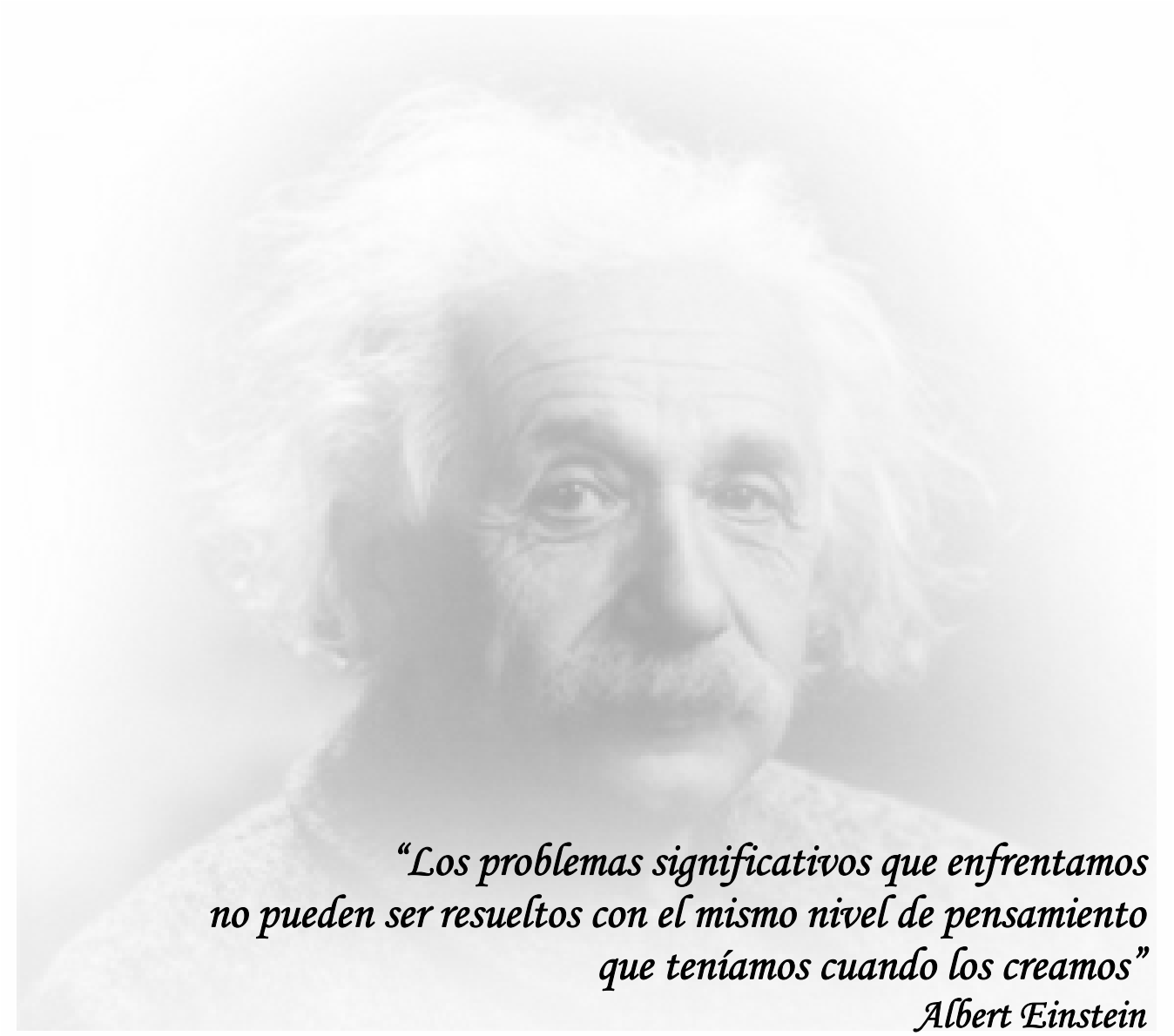
Tutores:
M.Sc. David Ardite Martínez

_____

Ing. Elián Cutiño Díaz

_____

PENSAMIENTO.



AGRADECIMIENTOS.

A los oficiales, profesores y estudiantes del grupo de desarrollo UCI_MINFAR que de una forma u otra nos han aportado informaciones importantes para el desarrollo de la investigación.

A Rolando por darnos las ideas y apoyo en los inicios de la realización de la Tesis.

A nuestros tutores y cotutor por estar presentes en todo momento de la Tesis, revisando y apoyando nuestras ideas.

A todos aquellos que de una forma u otra nos han aportado un grano de arena en nuestra formación personal y profesional.

A los que confiaron en nosotros y siempre nos apoyaron en las buenas y las malas.

A Fidel y a la Revolución.

DEDICATORIA.

A mis padres, Melba Pérez y Rudys Díaz por ser las personas más importante en mi vida, mis guías y mi fuente de inspiración.

A mis dos hermanos, que han estados en diferentes etapas de mi vida apoyándome y aconsejándome, Yizi y Rudys (Diovis), además de las personas especiales que los acompañan.

A mis sobrinos Danielito y Lizt Évelin por ser parte de mi corazón.

A mi familia que no por lejos están ausente, y a todos aquellos que se han comportado como tal.

A mis amigos de Secundaria y Pre, que aún mantienen comunicación y comparten conmigo.

A mis amigos de siempre, que durante cinco años de una larga carrera de estudios y vivencias humanas hemos compartido momentos inolvidables:

Giri, Anín, Yuliemis, Yaritza, Yusmilia, Daliagna, Licet, Yordan.

A las amigas del apto 67203, a mis compañeros de aula y en especial a mi compañero de Tesis y compañeros de proyecto.

A las personas que de una forma u otra siempre me han apoyado y se han preocupado en todo durante el desarrollo de mi carrera.

A todos, muchísimas gracias, ustedes forman parte de este logro.

Yiselis

DEDICATORIA.

A mis padres Enrique y Angelina.
A mi mamá, por inspirarme siempre a ser mejor cada día.
A mi padre, por ser mi ejemplo y guía.
A mis hermanos, por ser parte de los sentimientos de mi vida.
A mis queridos abuelos por ser ejemplos de sacrificios.
A mis Tías, por quererme tanto.
A mis familiares y a la memoria de los que ya no están.
A mis amigos de toda la vida, Jacqueline, yehilit, Yaimila, Orlando, Andy, Yudel, por estar siempre apoyándome y por compartir buenos y malos momentos.
A mis compañeros del grupo y en especial a mi compañera de tesis.
A aquellas personas que siempre creyeron en mí y a las que no creyeron también.
A los que me acompañaron en tantas horas de Trabajo.

Yaiser

RESUMEN.

El presente trabajo está enmarcado en los Tribunales Militares, donde actualmente funciona un sistema informático elaborado en 1993, sobre un sistema operativo y con recursos de programación propios del momento, jugando un importante papel en apoyo al trabajo judicial. En la actualidad, se hace necesaria la confección de una nueva *versión*, en un lenguaje de programación avanzado y en un ambiente de trabajo que permita la migración hacia sistemas de código abierto.

Con el propósito de resolver los problemas actuales, esta investigación tiene como objetivo el diseño de una arquitectura adecuada para el Sistema Informático de los Tribunales Militares de Región basada en una tecnología Web, en la misma se definen también los objetivos específicos y las tareas a resolver para la implementación del nuevo Sistema, además se tendrán en cuenta las experiencias acumuladas en la explotación del sistema actual, proyectando la futura organización estructural del mismo.

El éxito de la aplicación está en el resultado de la propuesta de una arquitectura, que les facilite un mejor desempeño de entendimiento y construcción a los demás miembros del equipo de desarrollo. Para esta se ha realizado una descripción de la propuesta, mediante la explicación de las diferentes características de los elementos de diseño que debe cumplir el Sistema para su desarrollo posterior.

PALABRAS CLAVE.

- Arquitectura de software.
- Estilos.
- Patrones.
- Diseño.
- Vistas Arquitectónicas.
- Componentes.

ÍNDICE.

INTRODUCCIÓN.

El desarrollo de las Tecnologías de la Información y las Comunicaciones (TICs) ha revolucionado y transformado la Economía Mundial, estas son consideradas la base para construir una Sociedad de la Información y una Economía del Conocimiento, que permitan a la vez generar nuevas oportunidades de acceso a la información, mejorar la productividad e impulsar el desarrollo.

Para alcanzar la sociedad de la información y el conocimiento, la aplicación masiva de las TICs, debe hacerse sobre un sistema socioeconómico que funcione y se base en la justicia, equidad social y en la solidaridad entre los hombres.

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

Durante los años de Revolución, el país se ha visto afectado fuertemente por el bloqueo económico que tiene impuesto los Estados Unidos. No obstante el Estado, ha buscado alternativas para seguir impulsando el desarrollo y ha decidido no quedarse atrás, por lo que ha ido poco a poco incorporando nuevos avances en el mundo de la Informática y las Telecomunicaciones y alternativas de planificación en todos los órdenes dentro de sus empresas y organismos; centrado en la idea de buscar los mejores mecanismos de planificación tanto en el orden financiero como en el material, encaminado al apoyo del proceso de toma de decisiones.

El Ministerio de las Fuerzas Armadas Revolucionarias (MINFAR) es uno de los órganos que contribuye a su propio desarrollo en las transformaciones de las nuevas tecnologías; desarrollando sistemas informáticos que faciliten el control de los procesos internos, para mejorar el trabajo de administración, planificación y toma de decisiones.

Dentro del MINFAR se encuentra la Dirección de Tribunales Militares (DTM), “órgano administrativo del Tribunal Supremo Popular al que corresponde llevar a cabo la dirección militar y organizativa, así como el control y la inspección de las funciones no jurisdiccionales de los Tribunales Militares” (TM) (Quesada, 2002), formando parte del Sistema de Tribunales de la República de Cuba y se rigen por lo establecido en la Constitución. (Quesada, 2002). Los Tribunales Militares en el ejercicio de sus funciones tienen como

objetivos específicos prevenir y reprimir, a través de sus pronunciamientos en los actos de justicia, todo hecho delictivo que afecte o pueda afectar la seguridad del Estado Cubano, la capacidad y disposición combativa de las instituciones armadas, la disciplina militar o el orden reglamentario establecido para el cumplimiento del Servicio Militar, así como las demás infracciones de la ley penal, cumplir y hacer cumplir la Legalidad Socialista y contribuir a la educación de los militares en la observancia estricta de las leyes, reglamentos, disposiciones y órdenes militares (Quesada, 2002).

Figura 1. Organización y estructura de los Tribunales Militares.

En los Tribunales Militares se encuentra en explotación el "Sistema Informático para los Tribunales Militares", elaborado en el año 1993, para un Sistema Operativo y con los recursos de programación propios del momento en que se desarrolló, lo cual jugó un importante papel en apoyo del trabajo judicial. Se hace necesaria la confección de una nueva *versión* en un lenguaje de programación y en un entorno actualizado, que permita la migración en un futuro hacia sistemas de códigos abierto. Además

perfeccionar su estructura, la interacción con el mismo e integrarle nuevas opciones de trabajo de acuerdo con las experiencias acumuladas en su explotación, de forma que no juegue solo un rol informativo, sino que tenga también un papel más activo en apoyo a la dirección de la actividad judicial, en los Tribunales Militares.

Las deficiencias que el mismo presenta, por las cuales no satisface las necesidades actuales de sus usuarios son las siguientes:

- No permite consultar la información por criterios de selección.
- Han surgido nuevas necesidades informativas que no se recogen en el sistema, y por tanto en su arquitectura.
- No tiene implementado sistema de seguridad alguno, referido a autentificación, auditoria, derechos de los usuarios a las opciones del mismo según la categoría o grupos a los que pertenecen.
- Está diseñado con los procesos de todos los niveles, sin ser necesario para usarlo en uno específico.
- No tiene implementado una ayuda en línea, ni elaborado un manual de explotación para facilitar el aprendizaje y uso del mismo.

Luego del estudio de las nuevas necesidades de procesamiento de la información en los Tribunales Militares y la solicitud de los clientes de realizar el nuevo sistema con una tecnología Web, para los procesos judiciales que se desarrollan en el nivel de Región, se realiza un análisis sobre las posibilidades y beneficios de construir el sistema basado en dicha tecnología.

Por su parte las aplicaciones Web ofrecen un contenido accesible para cualquier usuario desde cualquier dispositivo; facilita una gestión eficiente de la información, fomentando la interoperabilidad entre estaciones de trabajo; enriquece las aplicaciones para mejorar la usabilidad; así como la creación de infraestructuras para ofrecer una Web de confianza. Estos aspectos solo pueden alcanzarse mediante el desarrollo de elementos que intervienen en una arquitectura de software.

Después de verificar las ventajas que ofrece la realización de una aplicación basada en tecnologías Web y el papel que juega la definición de una arquitectura para su desarrollo, se ha definido para esta investigación el siguiente **problema científico**: ¿Cómo realizar el diseño arquitectónico del Sistema Informático de los Tribunales Militares de Región para tecnología Web?

Este trabajo enmarca su **objeto de estudio** en los procesos informativos de los Tribunales Militares de Región y su **campo de acción** es el diseño arquitectónico del Sistema Informático de los Tribunales Militares de Región.

Con el propósito de resolver el problema antes planteado se ha trazado el siguiente **objetivo general**: diseñar la arquitectura basada en una tecnología Web para el Sistema Informático de los Tribunales Militares de Región, con los siguientes **objetivos específicos:**

- Describir la arquitectura del sistema a través de las diferentes vistas que utiliza el arquitecto según la metodología utilizada.
- Determinar los estilos y patrones arquitectónicos que se emplearán durante el proceso de desarrollo y diseño del sistema.
- Crear la documentación de la Arquitectura del Sistema.

Como **tareas** que se llevarán acabo para darle cumplimiento a los objetivos trazados se tienen:

- Investigación del sistema existente en los Tribunales Militares.
- Estudio detallado de cómo se gestiona la información en los Tribunales Militares de la Región.
- Estudiar bibliografía referida a la arquitectura de Sistemas Informáticos.
- Investigación de los tipos de arquitecturas y patrones de diseño más factibles para mejorar el diseño del Sistema.
- Selección de herramientas que permitan describir la Arquitectura.
- Definición de la estrategia de trabajo.

Aportes prácticos esperados del trabajo.

Se espera obtener una arquitectura para el Sistema Informático de los Tribunales Militares de Región que sea modular, flexible, orientado a los requisitos del Sistema y que cumpla con los parámetros de eficiencia, funcionalidad y seguridad.

El presente trabajo consta de Introducción, tres capítulos, conclusiones, recomendaciones, bibliografías y un anexo.

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

Capítulo 2. Características de la Arquitectura del Sistema: se realiza una descripción del tipo de arquitectura que va a tener el Sistema de Tribunales Militares de Región.

Capítulo 3. Diseño de la Arquitectura: Se realiza la representación de la arquitectura mediante las 4+1 vistas definidas por RUP.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

1.1. Introducción.

En este capítulo se realiza una fundamentación de los aspectos a tratar en el resto del documento, se brinda una breve descripción de los estilos y patrones de diseños arquitectónicos existentes y más usados, así como la metodología, artefactos y herramientas a emplear para realizar la documentación de la arquitectura de un sistema.

A medida que aumenta la complejidad de los sistemas de software surgen nuevos aspectos del desarrollo de aplicaciones que hasta ese momento no se habían tenido en cuenta, al menos de una forma explícita. De este modo, el proceso de desarrollo se ha ido convirtiendo gradualmente en una labor de Ingeniería, en la que nuevas tareas, como la elaboración de especificaciones, el diseño del sistema, la construcción de prototipos, la integración y pruebas, la gestión de la configuración y otras muchas han ido cobrando importancia frente a las labores de programación que eran las que primaban en un inicio. La *Ingeniería de Software* ha ido respondiendo a esta situación con el desarrollo de nuevos modelos, notaciones, técnicas y métodos.

Dentro de esta tendencia se encuadra el creciente interés por los aspectos arquitectónicos del software del que se está siendo testigo en los últimos tiempos. Estos aspectos se refieren a todo lo relativo a la estructura de alto nivel de los sistemas: su organización en subsistemas y la relación entre estos, la construcción de aplicaciones vistas como una actividad fundamentalmente composicional en la que se reutilizan elementos creados posiblemente por terceros; el desarrollo de familias de productos caracterizadas por presentar una arquitectura común; el mantenimiento y la evolución entendidos como sustitución de unos *componentes* por otros dentro de un marco arquitectónico, etc. Un aspecto crítico a la hora de desarrollar sistemas de software complejos es el diseño de su arquitectura, representada como un conjunto de elementos computacionales y de datos interrelacionados de un modo determinado.

1.2. Elementos fundamentales de la Arquitectura de Software (AS).

1.2.1. Inicio de la Arquitectura del Software.

La Arquitectura de Software es uno de los grandes temas actuales que han dominado prácticamente toda la década del 90 junto a otros temas de Ingeniería, lo cual nos introduce en el hecho de que la misma forma parte de un contexto más amplio que es el de la Ingeniería de Software. Aunque en sus inicios no se le dio mucha importancia cuando (Dijkstra, David Parnas y Fred Brooks) [1] trataron de advertir la necesidad de su utilización, siendo los primeros en usar el término de Arquitectura de Software. La Arquitectura de Software fue tomando valor significativo a medida que los programadores se fueron dando cuenta de la importancia que representaba para la creación de un software, principalmente da una visión general del proyecto que se está desarrollando y permite comprender bien el sistema para resolver cualquier problema con solo analizar la arquitectura. Dentro de los grandes temas en la década de los noventa también se encuentra el surgimiento de los *patrones*[2], entre los más conocidos probablemente, están los patrones de diseño, propuestos por la llamada banda de los cuatros en 1995, (Erich Gamma, Richard Helm, Ralph Jonson, John Vlissides)[3] y por patrones más propiamente arquitectónicos, los publicado en 1996 por un conjunto de autores liderados por Frank Buschmann que se conocen también por la banda de los cinco (Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad , Michael Stal)[4]. Otro tema que dominó no tanto el siglo XX, sino lo que va del siglo XXI, son los llamados métodos ágiles o *heterodoxos*, el más conocido de ellos *eXtreme Programming*, luego vendrían *Scrum*, Evo, FDD, DSDM, RUP, AM, Crystal, LD, ASD, siendo estos otros procesos de desarrollo de software. Es importante destacar que la AS ha tomado y seguirá tomando auge, tanto que en la actualidad, el Proceso Unificado de Desarrollo (RUP)[5], metodología utilizada en la realización de un software usando como lenguaje de modelado a UML[6], tiene entre sus principales características que se centra en la arquitectura.(Reynoso, 2004)

[1] Científico de la computación, pionero temprano de la ingeniería de programas informáticos y un ingeniero de software además de ser científico de la computación.
[2] Ver sección 1.4.1.
[3] Personalidades de la especialidad que han contribuido con el desarrollo de los patrones de diseño.
[4] Personalidades de la especialidad que han contribuido con el desarrollo de los patrones propiamente de la arquitectura.
[5] Ver sección 1.5.1.
[6] Ver sección 1.5.2.

1.2.2. Necesidades de definir una Arquitectura.

Si se desea construir un Software de complejidad media como es el caso de los Sistemas de Tribunales Militares de Región o de mayor complejidad como es el caso de muchos sistemas en el mundo, se necesita tener una visión común de sistemas en desarrollo.

Lo importante es comprender que para la arquitectura existen varias perspectivas y que dentro de estas se pueden mostrar vistas dependiendo del enfoque que se quiera y que al diseñar una solución se debe pensar en ellas.

La Arquitectura de Software y el uso de patrones han sido de los temas que han dominado la década de los 90, dentro de la Ingeniería de Software como vía para entender y hacer progresar a los desarrolladores hasta tener una visión común del sistema que están desarrollando. Es decir se necesita una Arquitectura de Software para: (Ivar Jacobson, 2004)**.**

- Comprender el Sistema.
- Organizar el desarrollo.
- Fomentar la Reutilización.
- Hacer evolucionar el Sistema.

La necesidad de definir una arquitectura que tuviera en cuenta las visiones de otros interesados. Muchos ingenieros han diseñado los distintos mecanismos de arquitectura y se han plasmado en este capítulo de fundamentación teórica donde se hace referencia a la arquitectura que ha servido y sirve de mucho a todos los involucrados con el desarrollo de software.

1.2.3. ¿Qué es la Arquitectura de Software?

Aún cuando existen cientos de definiciones sobre qué es la Arquitectura de Software las ideas están centradas en las siguientes tres variantes:

1. La Arquitectura de Software como un proceso dentro del ciclo de vida de un sistema.
2. La Arquitectura de Software como la Topología del Sistema. Es decir la forma de articular los diferentes estilos y componentes dentro de una solución.
3. La Arquitectura de Software como una disciplina profesional y académica.

Dentro de las diferentes definiciones de arquitectura dichas por varias entidades y personalidades de la especialidad, se pueden encontrar:

"La Arquitectura de Software es la organización fundamental de un sistema encarnado en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución". (2007).

"… La arquitectura de un sistema constituye un amplio marco que describe su forma y su estructura, sus *componentes* y cómo estos encajan juntos..." (Pressman, 2005).

La más adoptada aún cuando es una definición muy básica, es la definida en PRESSMAN que plantea lo siguiente:

"… es una descripción de los subsistemas y los *componentes* de un sistema informático y las relaciones entre ellos (...)" (Pressman, 2005)

En su introducción a UML, Grady Booch, James Rumbaugh e Ivar Jacobson, específicamente refiriéndose a la metodología RUP han formulado un esquema de cinco vistas[7] interrelacionadas que conforman la arquitectura de software. En esta perspectiva, la Arquitectura de Software es un conjunto de decisiones significativas sobre:

1. La organización del sistema de software.
2. La selección de elementos estructurales y sus interfaces a través de los cuales se constituye el sistema.
3. Su comportamiento, según resultados de las colaboraciones entre esos elementos.
4. La composición de esos elementos estructurales y de comportamiento en subsistemas progresivamente mayores.
5. El estilo arquitectónico que guía esta organización: los elementos estáticos y dinámicos; sus interfaces, sus colaboraciones y su composición

En el desarrollo de una Arquitectura de Software para la implementación de un sistema de forma económica ahora y en el futuro se pone de manifiesto la interacción entre los *Casos de Uso* (CU) y la Arquitectura; es decir los CU son directores de la arquitectura y a su vez esta guía la realización de los CU. La Arquitectura se ve condicionada por los siguientes factores: (Ivar Jacobson, 2004)

- Sobre qué productos de software se desea desarrollar el sistema. (Sistema Operativo, Gestor de *Base de Datos*, etc.)
- El o los productos de *Middleware* que se quieren utilizar.
- Sistemas heredados a incorporar en el sistema.
- Los estándares y políticas corporativas.
- Los requisitos no funcionales generales.
- La distribución física de cada uno de los elementos del sistema.

Además en la Arquitectura influye la experiencia del equipo con respecto a este tema, así como los patrones seleccionados para su confección.

[7] Ver sección 2.4.

Llegar a una Arquitectura estable para el desarrollo de un Software es un proceso complejo y voluminoso que se desarrolla en iteraciones en la fase de elaboración del sistema donde intervienen el o los arquitectos y los desarrolladores del sistema, los cuales como resultados de su trabajo definen la Línea Base de la Arquitectura.

Al finalizar la fase de elaboración del sistema se han desarrollado diferentes modelos y artefactos que representan los diferentes factores que conducen, guían o condicionan la arquitectura, obteniéndose las primeras versiones de los modelos de Casos de Uso, Análisis, Diseño, Distribución, Implementación y Prueba. Esta agregación de modelos constituye la Línea Base de la Arquitectura.

Como se ha podido apreciar la Arquitectura de Software no tiene una sola dimensión, está formada por las múltiples vistas concurrentes siguientes: (Ivar Jacobson, 2004).

- La vista de Casos de Uso.
- La vista Lógica.
- La Vista de Componentes.
- La vista de Procesos.
- La vista de Despliegue.

Las mismas, serán explicadas más adelante una vez escogida la metodología para el desarrollo del Sistema y descripción de la Arquitectura.

1.3. Estilos Arquitectónicos*.*

Durante la definición de la arquitectura de software de un sistema están presentes las soluciones que se le darán a las diferentes problemáticas que enfrentará el equipo de desarrollo en la fase de construcción del sistema, muchas de estas soluciones constituyen patrones estándares que son reutilizados y ayudan a los ingenieros de software a realizar diseños mejores y más comprensibles. (Carlos Reynoso – Nicolás Kicillof)

La comunidad de patrones ha aplicado estas ideas para registrar soluciones estándares a problemas de la arquitectura que ocurren frecuentemente, los Estilos Arquitectónicos se encuentran en el centro de la arquitectura y constituyen buena parte de su sustancia. Los patrones de arquitectura están claramente dentro de la disciplina arquitectónica, solapándose con los estilos. Los partidarios de los estilos se definen desde el inicio como arquitectos. (Carlos Reynoso – Nicolás Kicillof).

Un estilo arquitectónico encapsula decisiones esenciales sobre los elementos arquitectónicos y enfatiza restricciones importantes de los elementos y sus relaciones posibles. (Carlos Reynoso – Nicolás Kicillof).

Es decir los estilos arquitectónicos comprenden los *componentes* (elementos), conectores, configuraciones y restricciones de las aplicaciones informáticas así como sus relaciones y comportamiento.

Existen relativamente pocos estilos arquitectónicos y en la actualidad no existe consenso para ser referenciado, aún cuando se conjugan o combinan varios estilos en una aplicación real.

Los estilos casi siempre se usan combinados; cada capa o componente puede ser internamente de un estilo diferente al de la totalidad; muchos estilos se encuentran ligados a dominios específicos o a líneas de producto particulares.

A continuación se describen de forma resumida algunos estilos, apenas los más representativos, vigentes y que de una forma u otra pudieran estar representados en el diseño de la arquitectura del Sistema de Tribunales Militares de Región y que son los más usados en la actualidad por muchos arquitectos de software en el Mundo; entre estos se encuentran:

1. Estilos de Flujo de Datos:
   - ❖ Tubería y filtros.
2. Estilos Centrados en Datos:
   - ❖ Arquitecturas de Pizarra o Repositorio.
3. Estilos de Llamada y Retorno:
   - ❖ Modelo-Vista-Controlador (MVC).
   - ❖ Arquitecturas en Capas.
   - ❖ Arquitecturas Orientadas a Objetos.
   - ❖ Arquitecturas Basadas en Componentes.
4. Estilos Peer-to-Peer (punto a punto):
   - ❖ Arquitecturas Basadas en Eventos.
   - ❖ Arquitecturas Orientadas a Servicios**.**

- Arquitecturas Basadas en Recursos.

1.3.1. Estilos de llamada y retorno.

Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Dentro de los miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objetos y los sistemas jerárquicos en capas.

- **Model-View-Controller (modelo-vista-controlador, MVC).**

Reconocido como estilo arquitectónico por Taylor y Medvidovic, el MVC ha sido propio de las aplicaciones en *Smalltalk* por lo menos desde 1992, antes que se generalizaran las arquitecturas en capas múltiples. En ocasiones se le define más bien como un patrón de diseño o como práctica recurrente y en estos términos es referido en el marco de la estrategia arquitectónica de *Microsoft*.

El patrón conocido como Modelo-Vista-Controlador separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

- ✓ Modelo: El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- ✓ Vista: Maneja la visualización de la información.
- ✓ Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

Tanto la vista como el controlador dependen del modelo. Esta separación permite construir y probar el modelo independientemente de la representación visual. La separación entre vista y controlador puede ser secundaria en aplicaciones de clientes ricos y de hecho, muchos frameworks (marcos de trabajo) de interfaz implementan ambos roles en un solo objeto. En aplicaciones Web, por otra parte, la separación entre la vista (el *browser, navegador*) y el controlador (los componentes del lado del servidor que manejan los requerimientos de *HTTP*) está mucho más definida.

Ventajas:

1. Soporte de vistas múltiples. Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Por ejemplo, múltiples páginas de una aplicación de Web pueden utilizar el mismo modelo de objetos, mostrado de maneras diferentes.
2. Adaptación al cambio. Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares o *PDAs*. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo. Este patrón sentó las bases para especializaciones ulteriores, tales como Page Controller (Página controladora) y Front Controller (Frente controlador).

Desventajas.

1. Complejidad. El patrón introduce nuevos niveles de *indirección* y por lo tanto aumenta ligeramente la complejidad de la solución. También se profundiza la orientación a eventos del código de la interfaz de usuario, que puede llegar a ser difícil de depurar. En rigor, la configuración basada en eventos de dicha interfaz corresponde a un estilo particular (arquitectura basada en eventos) que aquí se examina por separado.
2. Costo de actualizaciones frecuentes desacoplan el modelo de la vista, lo que no significa que los desarrolladores del modelo puedan ignorar la naturaleza de las vistas.
3. Si el modelo experimenta cambios frecuentes, por ejemplo, podría desbordar las vistas con una lluvia de requerimientos de actualización. Hace pocos años sucedía que algunas vistas, tales como las pantallas gráficas, involucraban más tiempo para plasmar el dibujo que el que demandaban los nuevos requerimientos de actualización.

- **Arquitecturas en Capas.**

Los sistemas o arquitecturas en capas constituyen uno de los estilos que aparecen con mayor frecuencia mencionados como categorías mayores del catálogo o por el contrario, como una de las posibles

encarnaciones de algún estilo más envolvente. (Garlan y Shaw)[8] definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.

La arquitectura por capas es un estilo de arquitectura en la que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño, un ejemplo básico de esto es separar la capa de datos de la capa de presentación al usuario.

La ventaja principal de este estilo, es que el desarrollo se puede llevar a cabo en varios niveles y en caso de algún cambio sólo se ataca al nivel requerido sin tener que revisar entre código mezclado.

Además permite distribuir el trabajo de creación de una aplicación por niveles, de este modo, cada grupo de trabajo está totalmente abstraído del resto de los niveles, simplemente es necesario conocer la *API* que existe entre niveles.

En el diseño de sistemas informáticos actual se suele usar las arquitecturas multinivel o Programación por capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

El diseño más en boga actualmente es el diseño en tres niveles (o en tres capas).

Capas o niveles.

1.- Capa de presentación: es la que ve el usuario, presenta el sistema al usuario, le comunica la información y captura la información del usuario dando un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio.

[8] Son personalidades que han influido en el desarrollo de Software, principalmente en la evolución de la arquitectura, tienen como nombre Mary Shaw y David Garlan, proporcionaron una sistematización iluminadora, explicando las diferencias entre definiciones en función de distintas clases de modelos.

2.- Capa de negocio: es donde residen los programas que se ejecutan, recibiendo las peticiones del usuario y enviando las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de *base de datos* para almacenar o recuperar datos de él.

3.- Capa de datos: es donde se ubican los datos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Todas estas capas pueden residir en un único ordenador (no sería lo normal), si bien lo más usual es que haya una multitud de ordenadores donde reside la capa de presentación (son los clientes de la arquitectura cliente/servidor). Las capas de negocio y de datos pueden residir en el mismo ordenador, y si el crecimiento de las necesidades lo aconseja se pueden separar en dos o más ordenadores. Así si el tamaño o complejidad de la base de datos aumenta, se puede separar en varios ordenadores los cuales recibirán las peticiones del ordenador en que resida la capa de negocio.

Si por el contrario fuese la complejidad en la capa de negocio lo que obligase a la separación, esta capa de negocio podría residir en uno o más ordenadores que realizarían solicitudes a una única base de datos. En sistemas muy complejos se llega a tener una serie de ordenadores sobre los cuales corre la capa de datos y otra serie de ordenadores sobre los cuales corre la base de datos.

En una arquitectura de tres niveles, los términos "capas" y "niveles" no significan lo mismo ni son similares. El término "capa" hace referencia a la forma como una solución es segmentada desde el punto de vista lógico: Presentación/ Lógica de Negocio/ Datos.

En cambio, el término "nivel", corresponde a la forma en que las capas lógicas se encuentran distribuidas de forma física. Por ejemplo:

- Una solución de tres capas (presentación, lógica, datos) que residen en un solo ordenador (Presentación+lógica+datos). Se dice, que la arquitectura de la solución es de tres capas y *un nivel*.

- Una solución de tres capas (presentación, lógica, datos) que residen en dos ordenadores (presentación+lógica, lógica+datos). Se dice que la arquitectura de la solución es de tres capas y *dos niveles*.
- Una solución de tres capas (presentación, lógica, datos) que residen en tres ordenadores (presentación, lógica, datos). La arquitectura que la define es: solución de tres capas y *tres niveles*.

Ventajas:

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

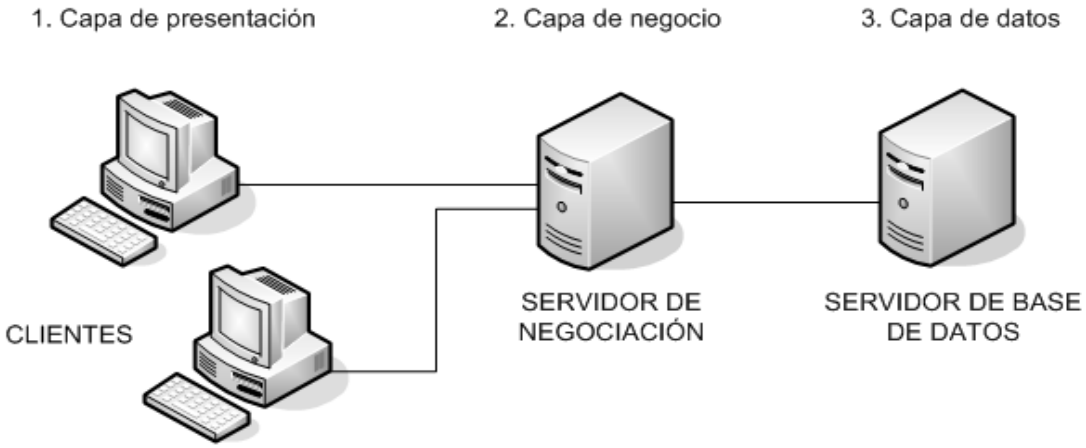


Figura 1.1. Arquitectura en Capas.

Desventajas:

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

- **Arquitecturas Orientadas a Objetos (OO).**

Nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. En la caracterización clásica de David Garlan y Mary Shaw, los objetos representan una clase de componentes que ellos llaman managers (gestoras*)*, debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos. En la semblanza de estos autores curiosamente no se establece como cuestión definitoria el principio de *herencia*. Ellos piensan que a pesar de que la relación de herencia es un mecanismo organizador importante para definir los tipos de objeto en un sistema concreto, ella no posee una función arquitectónica directa. En particular, en dicha concepción la relación de herencia no puede concebirse como un conector, puesto que no define la interacción entre los componentes de un sistema. Además, en un escenario arquitectónico la herencia de propiedades no se restringe a los tipos de objeto, sino que puede incluir conectores e incluso estilos arquitectónicos enteros.

Resumiendo las características de las arquitecturas OO, se podría decir que:

- Los componentes del estilo se basan en principios OO: *encapsulamiento*, *herencia* y *polimorfismo*. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación.
- Las interfaces están separadas de las implementaciones. En general la distribución de objetos es transparente, y en el estado de arte de la tecnología (lo mismo que para los componentes en el sentido de que apenas importa si los objetos son locales o remotos. El mejor ejemplo de OO para sistemas distribuidos es Common Object Request Broker Architecture, Arquitectura Común de Intermediarios en Peticiones a Objetos (CORBA), en la cual las interfaces se definen mediante Interface Description Language, Lenguaje de Especificación de Interfaces (IDL); un Object Request Broker media la interacción entre objetos clientes y objetos servidores en ambientes distribuidos.
- En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. En muchos componentes, los objetos interactúan a través de invocaciones de funciones y procedimientos. Hay muchas variantes del estilo; algunos sistemas, por ejemplo, admiten que los objetos sean tareas concurrentes; otros permiten que los objetos posean múltiples interfaces.

Se puede considerar el estilo como perteneciente a una familia arquitectónica más amplia, que algunos autores llaman Arquitecturas de Llamada-y-Retorno (Call-and-Return). Desde este punto de vista, sumando las *APIs* clásicas, los componentes (en el sentido Component Object Model, Modelo de Objeto de Componente (COM) y JavaBeans*)* y los objetos.

Ventajas:

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

Desventajas:

- Para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad.
- Efectos colaterales en cascada: si A usa B y C también lo usa, el efecto de C sobre B puede afectar A.

Entre las limitaciones, el principal problema del estilo se manifiesta en el hecho de que para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad. Esta situación contrasta con lo que es el caso en estilos tubería-filtros, donde los filtros no necesitan poseer información sobre los otros filtros que constituyen el sistema. La consecuencia inmediata de esta característica es que cuando se modifica un objeto (por ejemplo, se cambia el nombre de un método, o el tipo de dato de algún argumento de invocación) se deben modificar también todos los objetos que lo invocan.

En la literatura sobre estilos, las arquitecturas orientadas a objetos han sido clasificadas de formas diferentes, conforme a los diferentes puntos de vista que alternativamente enfatizan la jerarquía de componentes, su distribución topológica o las variedades de conectores.

- **Arquitecturas Basadas en Componentes.**

Esta arquitectura se considera una evolución de las tecnologías de orientación a objeto que comenzó hacia el año 1994.

Hoy en día existe un buen número de definiciones de componentes, pero Clemens Alden Szyperski proporciona una que es bastante operativa: un componente de software es una unidad de composición con interfaces especificadas contractualmente y dependencias del contexto explícitas. Que sea una unidad de composición y no de construcción quiere decir que no es preciso confeccionarla: se puede comprar hecha, o se puede producir en casa para que otras aplicaciones de la empresa la utilicen en sus propias composiciones. Pragmáticamente se puede también definir un componente como un artefacto diseñado y desarrollado de acuerdo con un estilo dado.

Los componentes en el sentido estilístico son las unidades de modelado, diseño e implementación.

Las interfaces están separadas de las implementaciones, y las interfaces y sus interacciones son el centro del diseño arquitectónico. Los componentes soportan algún régimen de abstracción, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución.
En cuanto a las restricciones, puede admitirse que una interfaz sea implementada por múltiples componentes. Usualmente, los estados de un componente no son accesibles desde el exterior. Que los componentes sean locales o distribuidos es transparente en la tecnología actual.
El marco arquitectónico estándar para la tecnología de componentes está constituido por los cinco puntos de vista de RM-ODP[9] (Empresa, Información, Computación, Ingeniería y Tecnología). La evaluación dominante del estilo de componentes subraya su mayor versatilidad respecto del modelo de objetos, Las tecnologías de componentes del período de inmadurez, asimismo, se consideraban afectadas por problemas de incompatibilidad de versiones e inestabilidad que ya han sido largamente superados en toda la industria.

1.3.2. Estilos Peer-to-Peer (Punto a Punto).

Esta familia, también llamada de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente. Los mensajes pueden ser enviados a componentes nominados o propalados mediante *broadcast*[10]. Miembros de la familia son los estilos basados en eventos, en mensajes, en servicios y en recursos.

- **Arquitecturas Orientadas a Servicios (SOA).**

Estas arquitecturas llamada SOA han recibido tratamiento intensivo en el campo de exploración de los estilos y se visualizan como una tendencia que habrá de ser dominante. Ahora bien, este predominio no se funda en la idea de servicios en general, comunicados de cualquier manera, sino que más específicamente va de la mano de la expansión de los *Web Services* basados en *XML*, en los cuales los formatos de intercambio se basan en XML y el protocolo de elección es *SOAP*. SOAP significa un formato

[9] Reference Model for Open Distributed Processing, ISO, Modelo de Referencia para Abrir el Proceso Distribuido, estándar de ISO.
[10] En castellano es difusión, es un modo de transmisión de información donde un nodo emisor envía información a una multitud de nodos receptores de manera simultánea, sin necesidad de reproducir la misma transmisión nodo por nodo.

de mensajes que es XML, comunicado sobre un transporte que por defecto es *HTTP*, pero que puede ser también HTTPs, *SMTP*, *FTP*, o casi cualquier otro.

Un Web Service es un sistema de software diseñado para soportar interacción máquina-a-máquina sobre una red. Posee una interfaz descripta en un formato procesable por máquina (específicamente *WSDL*). Otros sistemas interactúan con el Web Service de una manera prescripta por su descripción utilizando mensajes SOAP, típicamente transportados usando HTTP con una serialización en XML en conjunción con otros estándares relacionados a la Web.

En la literatura clásica referida a estilos, las arquitecturas basadas en servicios podrían engranar con lo que Garlan & Shaw definen como el estilo de procesos distribuidos. Otros autores hablan de Arquitecturas de Componentes Independientes que se comunican a través de mensajes.

Vale la pena destacar la forma en la cual el estilo redefine los viejos modelos propios de las arquitecturas orientadas a objetos y componentes, y al hacerlo, establece un modelo en el que es casi razonable pensar que cualquier entidad computacional (nativamente o mediando un wrapper, la envoltura) podría llegar a conversar o a integrarse con cualquier otra una vez resueltas las inevitables coordinaciones de filosofía.

Lo que hace diferentes a los Web Services de otros mecanismos de *RPC* como RMI, CORBA o DCOM es que utiliza estándares de Web para los formatos de datos y los protocolos de aplicación. Esto no sólo es un factor de corrección política, sino que permite que las aplicaciones ínter operen con mayor libertad, dado que las organizaciones ya seguramente cuentan con una infraestructura activa de HTTP y pueden implementar tratamiento de XML y SOAP en casi cualquier lenguaje y plataforma, ya sea descargando un par de equipos, adquiriendo el paquete de lenguaje o biblioteca que proporcione la funcionalidad o programándolo a mano. Además la descripción, publicación, descubrimiento, localización e invocación de los Web Services se puede hacer en tiempo de ejecución, de modo que los servicios que interactúan pueden figurarse la forma de operar de sus contrapartes, sin haber sido diseñados específicamente caso por caso. Por primera vez, este dinamismo es plenamente viable.

Desde el punto de vista arquitectónico, se puede hacer ahora una caracterización de las características del estilo:

- Un servicio es una entidad de software que encapsula funcionalidad de negocios y proporciona dicha funcionalidad a otras entidades a través de interfaces públicas bien definidas.
- Los componentes del estilo (o sea los servicios) están débilmente acoplados. El servicio puede recibir requerimientos de cualquier origen. La funcionalidad del servicio se puede ampliar o modificar sin rendir cuentas a quienes lo requieran. Los servicios son las unidades de implementación, diseño e implementación.
- Los componentes que requieran un servicio pueden descubrirlo y utilizarlo dinámicamente mediante *UDDI* y sus estándares sucesores. En general (aunque hay alternativas) no se mantiene persistencia de estado y tampoco se pretende que un servicio recuerde nada entre un requerimiento y el siguiente.

Como todos los otros estilos, las SOA poseen ventajas y desventajas. Como se trata de una tecnología que está en su pico de expansión, virtudes y defectos están variando.

Pero SOA es una manera de diseñar los sistemas de software y su ambiente, para permitir que los servicios puedan ser provistos a aplicaciones de usuarios finales, procesos de negocio “ejecutables”, o bien, otros servicios; a través de la publicación y descubrimiento de la interfaz de los servicios.

Dentro de los beneficios que este estilo brinda se encuentran:

1. Reusabilidad de Servicios:

- Disminución de tiempos y costos de desarrollo de las aplicaciones al utilizar servicios disponibles ya desarrollados, para resolver problemáticas comunes a otras aplicaciones.
- Disminución del riesgo al utilizar algo ya probado (robusto).
- Logro de mejoras globales en las aplicaciones al mejorar un servicio que está siendo usado por varias aplicaciones, agilidad frente al cambio.
- Disminución de los costos de administración y operación.

2. Interoperabilidad de Aplicaciones:

- Disminución de la complejidad de interrelación entre aplicaciones al utilizar un "protocolo común" de entendimiento, en sí una interfaz común.
- Disminución de la complejidad de integración, al interactuar con un elemento que se abstrae de la tecnología y ubicación de los servicios.

3. Utilización de un Medio Único de Acceso a Servicios:

- Eliminar la problemática de las integraciones "Punto a Punto", posibilitando la gobernabilidad de la integración entre aplicaciones.
- Facilita la administración, operación, auditoria y trazabilidad, en definitiva, la gobernabilidad y control sobre la integración de aplicaciones y su continuidad operacional.

1.4. Patrones.

Durante el desarrollo o elaboración de software al igual que la construcción de un edificio pueden sistematizarse principios y prácticas que constituyen soluciones a problemas comunes que ocurren en un mismo entorno. El Arquitecto Christopher Alexander definió las ideas sobre los "Lenguajes de Patrones", motivando a la comunidad de la Orientación a Objeto a definir, coleccionar y probar una gran cantidad y variedad de patrones de Software. La "Comunidad de Patrones" define un patrón como una solución a un problema que aparece frecuentemente. Es decir un patrón es una descripción de un problema y su solución que recibe un nombre y que puede emplearse en otros contextos.

La dinámica incontenible de la producción de patrones en la práctica de la arquitectura de software, han atenuado la idea de que los patrones de diseño constituyen sólo uno de los *paradigmas*, marcos o formas del diseño arquitectónico, cada uno de los cuales posee una historia y una fundamentación distinta, y presenta, como todas las cosas en este terreno, sus características, sus beneficios y sus limitaciones. Existen diversas clases de patrones: de análisis, de arquitectura (divididos en progresivamente estructurales, sistemas distribuidos, sistemas interactivos, sistemas adaptables), de diseño (conductuales, creacionales, estructurales), de organización o proceso, de programación y los llamados idiomas, entre otros.

Aún cuando existe una notoria relación entre todos los tipos de Patrones por su grado de madurez, adaptabilidad e importancia inciden más directamente en la arquitectura los Patrones de Arquitectura y los Patrones de Diseño.

1.4.1. Patrones de Arquitectura.

Según lo expresado en el texto de Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad y Michael Stal, Pattern-Oriented Software Architecture, Arquitectura de Software Orientada a Patrones (POSA) los patrones arquitectónicos son lo mismo que los estilos y ambos términos se usan de manera indistinta. En POSA los patrones arquitectónicos “expresan un esquema de organización estructural para los sistemas de software, proporcionan un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y lineamientos para organizar la relación entre ellos”.

Estrategias de diseño de Arquitectura:

Para definir qué patrones de arquitectura seleccionar entre la infinidad de patrones existentes, para aplicarlo al sistema se hace necesario trazar una estrategia a seguir.

Existen un conjunto de estrategias de diseño propuestas no necesariamente excluyentes para conformar un plano del sistema en desarrollo entre las que se pueden mencionar las siguientes:

1. Diseño arquitectónico basado en artefactos. Incluye modalidades bien conocidas de diseño orientado a objetos, tales como el OMT de Rumbaugh y el de Booch. En OMT, que puede considerarse representativo de la clase, la metodología de diseño se divide en tres fases, que son Análisis, Diseño del Sistema y Diseño de Objetos. En la fase de análisis se aplican tres técnicas de modelado que son modelado de objetos, modelado dinámico y modelado funcional. En la fase de diseño de sistema tienen especial papel lo que Rumbaugh llama implementación de control de software y la adopción de un marco de referencia arquitectónico, punto en el que se reconoce la existencia de varios prototipos que permiten ahorrar esfuerzos o se pueden tomar como puntos de partida. Algunos de esos marcos de referencia se refieren con nombres tales como transformaciones por lotes, transformaciones continuas, interfaz interactiva, simulación dinámica, sistema en tiempo real y administrador de transacciones. No cuesta mucho encontrar la analogía entre dichos marcos y los estilos arquitectónicos, concepto que en esa época todavía no había hecho su aparición. En el señalamiento de las ventajas del uso de un marco preexistente

también puede verse un reflejo de la idea de patrón, una categoría que no aparece jamás en todo el marco de la OMT, aunque ya había sido propuesta por el arquitecto británico Christopher Alexander varios años antes en 1977.

2. Diseño arquitectónico basado en casos de uso*.* Un caso de uso se define como una secuencia de acciones que el sistema proporciona para los actores. Los actores representan roles externos con los que el sistema debe interactuar. Los actores junto con los casos de uso, forman el modelo de casos de uso. Este se define como un modelo de las funciones que deberá cumplir el sistema y de su entorno, y sirve como una especie de contrato entre el cliente y los desarrolladores. El Proceso Unificado de Desarrollo (RUP), aplicando una arquitectura orientada por casos de uso. RUP definen el contenido estático del proceso y describen el proceso en términos de actividades, operadores y artefactos. La organización del proceso en el tiempo se define en fases. De acuerdo con el punto de vista, el concepto de estilo puede caer en diversas coordenadas del modelo, en las cercanías de las fases y los modelos de análisis y diseño. Es bueno señalar que el concepto de patrón responde con naturalidad a un diseño orientado a objetos. La estrategia de diseño basada en casos de uso, es dominada ampliamente por la orientación a objetos (tantos en los modelos de alto nivel como en la implementación) y por notaciones ligadas a UML, está experimentando reformulaciones y extensiones a fin de acomodarse a configuraciones arquitectónicas que no están estrictamente articuladas como objetos (basadas en servicios, por ejemplo), así como a conceptos de descripción arquitectónica y estilos.

3. Diseño arquitectónico basado en línea de producto*.* Comprende un conjunto de productos que comparten una colección de rasgos que satisfacen las necesidades de un determinado mercado o área de actividad. En la estrategia de arquitectura de *Microsoft*, este modelo está soportado por un largo conjunto de lineamientos, herramientas y patrones arquitectónicos específicos, incluyendo patrones y modelos .NET para aplicaciones de línea de negocios, modelos aplicativos en capas como arquitecturas de referencia para industrias, etc.

4. Diseño arquitectónico basado en dominios*.* Se considera una extensión del Diseño arquitectónico basado en línea de producto, se origina en una fase de análisis de dominio que, puede ser definido como la identificación, la captura y la organización del conocimiento sobre el dominio del problema, con el objetivo de hacerlo reutilizable en la creación de nuevos sistemas. El modelo del dominio se puede representar mediante distintas formas de representación bien conocidas en ingeniería del conocimiento,

tales como clases, diagramas de entidad-relación, redes semánticas y reglas. Relacionado con este paradigma se encuentra la llamada arquitectura de software específica de dominio, se puede considerar como una arquitectura de múltiples vistas, que deriva una descripción.

5. Diseño arquitectónico basado en patrones. Las ideas de Christopher Alexander sobre lenguajes de patrones han sido masivamente adoptadas y han conducido a la actual revolución por los patrones de diseño, sobre todo a partir del impulso que le confirieron las propuestas de la llamada "Banda de los Cuatro". Los patrones de diseño de software propuestos por la Banda buscan codificar y hacer reutilizables un conjunto de principios a fin de diseñar aplicaciones de alta calidad. Los patrones de diseño se aplican en principio sólo en la fase de diseño, aunque a la larga se han definido y aplicado patrones en las restantes etapas de desarrollo.

1.4.2. Patrones de Diseño.

Si bien hasta el momento no se ha enfatizado en la importancia del Análisis y Diseño orientados a objetos para el caso de los patrones de diseño son prácticamente imposible de utilizar bajo otras filosofías puesto que centran fortaleza en:

- ¿Cómo asignar las responsabilidades a los objetos?
- ¿Qué papel desempeña cada clase?
- ¿Cómo interactúan estos objetos?

Desde esta perspectiva dirigida por modelos, se definirán patrones como una plantilla de colaboración. Por tanto, se consideran los patrones de diseño como una colaboración entre clases e instancias, como su comportamiento [Jacobson, Booch, Rumbaugh].

En la fase de diseño de un Software los diagramas de interacción describen gráficamente la solución, a partir de los objetos en interacción, que las responsabilidades satisfacen.
Dentro de los Patrones de diseño más utilizados se encuentran los Patrones *GRASP* (General Responsability Asignment Software Patterns) que describen los principios fundamentales de la asignación de responsabilidades a objetos y que a continuación se relacionan:
Patrones GRASP

1. Experto.
2. Creador.

3. Controlador.
4. Bajo acoplamiento.
5. Alta Cohesión.
6. *Polimorfismo.*
7. Fabricación Pura.
8. *Indirección.*
9. “No hables con extraños”.

Existen otros Patrones de Diseño que se enmarcan dentro de los llamados Patrones de Diseño Estructurales, Creacionales y de Comportamientos como son los Patrones GoF (Gans of Four, Grupo de los Cuatro), que a continuación se relacionan solo algunos ya que los mismos son 23:

- Patrón GoF. Fachada.
- Patrón GoF. Singleton.
- Patrón GoF.Observer.

1.5. Tecnologías actuales a considerar.

1.5.1. Proceso de Desarrollo de Software.

¿Qué metodología se debe usar para el desarrollo de una Arquitectura de Software?

Para desarrollar un software, en un momento determinado, se debe definir una metodología. Todo desarrollo de software es riesgoso y difícil de controlar, pero si no se utiliza una metodología, lo que se obtiene son clientes insatisfechos con el resultado y desarrolladores aún más insatisfechos.

Lo que se hace con los proyectos pequeños de dos o tres meses, es separar rápidamente el aplicativo en procesos, cada proceso en funciones, y por cada función determinar un tiempo aproximado de desarrollo. Cuando los proyectos que se van a desarrollar son de mayor envergadura, ahí si toma sentido, una metodología de desarrollo, y se comienza a buscar cual sería la más apropiada para el caso de estudio. Lo cierto es que muchas veces no se encuentra la más adecuada y se termina por hacer o diseñar una propia metodología, algo que por supuesto no está mal, siempre y cuando cumpla con el objetivo.

A continuación se mencionarán tres de de las metodologías utilizadas en el proceso de desarrollo de un software, desde el punto de vista que permita describir la Arquitectura, estas son:

Proceso Unificado de Desarrollo (RUP), Programación Extrema (XP) y *Microsoft* Solution Framework (MSF).

Proceso Unificado de Desarrollo (RUP).

El Proceso Unificado de Desarrollo, RUP, es una metodología para el desarrollo de software orientados a objetos. Es un proceso de desarrollo de software, definido como un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software. Sin embargo, el proceso unificado es más que un proceso de trabajo, es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organizaciones y diferentes niveles de aptitud. Está constituido por 5 flujos de trabajo fundamentales: requisitos, análisis, diseño, implementación y prueba, los cuales tienen lugar sobre 4 etapas o fases: inicio, elaboración, construcción y transición. Esta metodología es adaptable para proyectos a largo plazo y establece refinamientos sucesivos de una arquitectura ejecutable.

Características específicas de RUP:

- Dirigido por casos de uso**:** Esto significa que el proceso de desarrollo sigue una trayectoria que avanza a través de los flujos de trabajo generados por los casos de uso. Los casos de uso se especifican y diseñan al principio de cada iteración, y son la fuente a partir de la cual los ingenieros de prueba construyen sus casos de prueba. Estos describen la funcionalidad total del sistema.
- **Centrado en la arquitectura:** Los casos de uso guían a la arquitectura del sistema y ésta influye en la selección de los casos de uso. La arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por las plataformas de software, sistemas operativos, sistemas de gestión de bases de datos, además de otros como sistemas heredados y requerimientos no funcionales.
- Iterativo e incremental **:** RUP divide el proceso en cuatro fases , dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y las cuales se definen según el nivel de madurez que alcanzan los productos que se van obteniendo con cada actividad ejecutada. La terminación de cada fase ocurre en el hito correspondiente a cada una, donde se evalúa que se hayan cumplido los objetivos de la fase en cuestión.

RUP está basado en componentes y utiliza UML (Lenguaje de Modelado Unificado (Unified Modeling Language) para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software.

Programación Extrema (XP).

La Programación Extrema, es una metodología ligera de desarrollo de software que se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. La metodología consiste en una programación rápida o extrema, utilizadas para proyectos de corto plazo.

Características de XP, la metodología se basa en:

- Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que se puede adelantar en algo hacia el futuro, se pueden hacer pruebas de las fallas que pudieran ocurrir. Es como si se obtuvieran los posibles errores.
- Refabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa.

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

- Los diseñadores y programadores se comunican efectivamente con el cliente y entre ellos mismos.
- Los diseños del software se mantienen sencillos y libres de complejidad o pretensiones excesivas.
- Se obtiene retroalimentación de usuarios y clientes desde el primer día gracias a las baterías de pruebas.
- El software es liberado en entregas frecuentes tan pronto como sea posible.
- Los cambios se implementan rápidamente tal y como fueron sugeridos.

- Las metas en características, tiempos y costos son reajustadas, permanentemente en función del avance real obtenido.

¿Qué es lo que propone XP?

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva del proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.
- El cliente o el usuario se convierte en miembro del equipo. (Escribano, 2002)

Microsoft Solution Framework (MSF).

MSF tiene las siguientes características:

- Adaptable: es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- Escalable: puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.
- Flexible: es utilizada en el ambiente de desarrollo de cualquier cliente.
- Tecnología Agnóstica: porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el modelo de Aplicación.

- Modelo de Arquitectura del Proyecto*:* Diseñado para acortar la planificación del ciclo de vida. Este modelo define las pautas para construir proyectos empresariales a través del lanzamiento de versiones.

- Modelo de Equipo*:* Este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles.
- Modelo de Proceso*:* Diseñado para mejorar el control del proyecto, minimizando el riesgo, y aumentar la calidad acortando el tiempo de entrega. Proporciona una estructura de pautas a seguir en el ciclo de vida del proyecto, describiendo las fases, las actividades, la liberación de versiones y explicando su relación con el Modelo de equipo.
- Modelo de Gestión del Riesgo: Diseñado para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.
- Modelo de Diseño del Proceso: Diseñado para distinguir entre los objetivos empresariales y las necesidades del usuario. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible a través de un enfoque iterativo. Las fases de diseño conceptual, lógico y físico proveen tres perspectivas diferentes para los tres tipos de roles: los usuarios, el equipo y los desarrolladores.
- Modelo de Aplicación: Diseñado para mejorar el desarrollo, el mantenimiento y el soporte, proporciona un modelo de tres niveles para diseñar y desarrollar aplicaciones software. Los servicios utilizados en este modelo son escalables, y pueden ser usados en un solo ordenador o incluso en varios servidores.

Visión general del MSF:

Fase 1 Estrategia y alcance:

- Elaboración y aprobación del documento de alcances del proyecto.
- Formación del equipo de trabajo y distribución de competencias y responsabilidades.
- Elaboración del plan de trabajo.
- Elaboración de la matriz de riesgos y plan de contingencia.

Fase 2 Planificación y prueba de concepto:

- Documento de planificación y diseño de arquitectura.
- Documento de plan de laboratorio (son las pruebas de conceptos)

Fase 3 Estabilización:

- Selección del entorno de pruebas piloto.
- Gestión de incidencias.
- Revisión de la documentación final de la arquitectura.
- Elaboración de plan de despliegue.
- Elaboración del plan de formación.

Fase 4 Despliegue:

- Registro de mejoras y sugerencias.
- Revisión de las guías y manuales de usuario.
- Entrega del proyecto y cierre del mismo.

Para concluir después de haber explicado las diferentes metodologías de desarrollo de software, se plantea que lo más importante antes de elegir la metodología que se usará para la implementación del software, es determinar el alcance que tendrá y luego determinar cuál es la que más se adapta a la aplicación, dicha por muchos desarrolladores de software.

1.5.2. Herramientas a considerar.

Primeramente se explicarán los lenguajes de modelado que históricamente se han utilizado para modelar Arquitectura dentro de esto, se encuentran:

- **El Lenguajes de Descripción Arquitectónica (ADLs).**

Los lenguajes de descripción de arquitecturas, o ADLs, ocupan una parte importante del trabajo arquitectónico desde la fundación de la disciplina. Se trata de un conjunto de propuestas de variado nivel de rigurosidad, casi todas ellas de extracción académica, que fueron surgiendo desde comienzos de la década de 1990 hasta la actualidad, más o menos en contemporaneidad con el proyecto de unificación de los lenguajes de modelado bajo la forma de UML. Los ADL difiere sustancialmente de UML, que al menos en su *versión* 1.x se estima inadecuado en su capacidad para expresar conectores en particular y en su modelo semántico en general para las clases de descripción y análisis que se requieren. Los ADLs permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones

que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento.

Los ADLs son bien conocidos en los estudios universitarios de Arquitectura de Software, pero muy pocos arquitectos de industria parecen conocerlos y son menos aún quienes los utilizan como instrumento en el diseño arquitectónico de sus proyectos. Hay unos veinte ADLs de primera magnitud y tal vez unos cuarenta o cincuenta propuestos en ponencias que no han resistido el paso del tiempo o que no han encontrado su camino en el mercado.

- **UML *(*Lenguaje Unificado de Modelado, Unified Modeling Language).**

A mediados de los noventa existían muchos métodos de análisis y diseño Orientado a Objetos lo que suponía que los mismos conceptos tenían distinta notación según el método de que se tratara. Ante esta situación de confusión, en 1994 Booch, Rumbaugh y Jacobson decidieron unificar sus métodos dando lugar a UML. Esta unificación fue promovida por el OMG de tal manera que UML se convirtió en la notación estándar para la descripción de métodos software. Según su definición, UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software, desde una perspectiva Orientada a Objetos. (RODRIGUEZ, 2006)

Luego de haber explicado los diferentes lenguajes posibles a utilizar en el modelado de un software, se especificarán las Herramientas CASE, donde una de las características de la arquitectura, es definir las herramientas a usar para el sistema, por ejemplo se mencionarán las herramientas que se utilizan históricamente, pero antes se define que es una herramienta CASE y sus ventajas:

- **Herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador).**

"Se puede definir a una herramienta CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software." (Hernández, 2004-2005)

Ventajas con la utilización de las herramientas CASE:

- ✓ Permiten el incremento en la velocidad de desarrollo de los sistemas.

- ✓ Permiten a los analistas tener más tiempo para el análisis y diseño y minimizar el tiempo para codificar y probar.
- ✓ En las etapas del proceso de desarrollo de software permiten:
  - ❖ Automatizar el dibujo de diagramas.
  - ❖ Ayudar en la documentación del sistema.
  - ❖ Ayudar en la creación de relaciones en la *base de datos*.
  - ❖ Generar estructuras de código.
- ✓ Aumentan la productividad. Esto se consigue a través de la automatización de determinadas tareas, como la generación de código y la reutilización de objetos o módulos.

Tipos de Herramientas CASE:

- ❖ **Rational Rose Enterprise Suite.**

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

- ❖ **Visual paradigm – UML.**

Visual Paradigm para UML es una de las herramientas UML CASE del mercado, considerada como muy completa y fácil de usar, con soporte multiplataforma y que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. Fue creada para el ciclo vital completo del desarrollo del software que lo automatiza y acelera, permitiendo la captura de requisitos, análisis, diseño e implementación. Visual Paradigm-UML también proporciona características tales como generación del código, ingeniería inversa y generación de informes. Tiene la capacidad de crear el esquema de clases a partir de una *base de datos* y crear la definición de base de datos a partir del esquema de clases. Permite invertir código

fuente de programas, archivos ejecutables y binarios en modelos UML al instante, creando de manera simple toda la documentación. Está diseñada para usuarios interesados en sistemas de software de gran escala con el uso del acercamiento orientado a objeto, además apoya los estándares más recientes de las notaciones de Java y de UML. Incorpora el soporte para trabajo en equipo, que permite que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios hechos por sus compañeros.

Una vez analizadas las tecnologías que son posibles para el desarrollo de un Software, y una pequeña comparación se ha decidido escoger como proceso de desarrollo a RUP, no solo por estar plasmada dentro de las normativas para el desarrollo de Software en las FAR sino, por tener como una de las principales características, que está centrada en la arquitectura, además de estar compuesta por diversos flujos de trabajo donde la captura de requisitos es fundamental para la descripción de la Arquitectura de un Sistema y como herramienta CASE se utilizara Visual Paradigm – UML por tener soporte multiplataforma, proporciona características tales como generación del código, ingeniería inversa y generación de informes, además tiene la capacidad de crear el esquema de clases a partir de una *base de datos* y crear la definición de base de datos a partir del esquema de clases, facilitando el trabajo a los demás desarrolladores y además que utiliza un lenguaje de modelado UML.

1.6. Artefactos que genera el Arquitecto de Software según RUP.

Un proceso de desarrollo de software define quién hace qué cómo y cuándo. RUP como metodología define cuatro elementos, los roles, que responden a la pregunta ¿Quién?, las actividades que responden a la pregunta ¿Cómo?, los flujos de trabajo de las disciplinas que responde a la pregunta ¿Cuándo?, y los productos, que responden a la pregunta ¿Qué?

Este últimos es el que a da lugar a los artefactos que son un trozo de información que es producido, modificado o usado durante el proceso de desarrollo de software. Los productos son los resultados tangibles del proyecto, las cosas que va creando y usando hasta obtener el producto final.

Un artefacto puede ser cualquiera de los siguientes:

- Un documento, como el documento de la arquitectura del software.
- Un modelo, como el modelo de Casos de Uso (CU) o el modelo de diseño.
- Un elemento del modelo, un elemento que pertenece a un modelo como una clase, un Caso de

Uso o un subsistema.

Según RUP la arquitectura se desarrolla en todas sus fases pero se tiene una arquitectura más robusta en las fases finales del proyecto. En las fases iniciales lo que se hace es ir consolidando la arquitectura por medio de *baselines* y se va modificando dependiendo de las necesidades del proyecto.
Al final de la fase de elaboración se obtiene una *baseline* de la arquitectura donde fueron seleccionados una serie de Casos de Uso Arquitectónicamente Relevantes (aquellos que ayudan a mitigar los riesgos más importantes, aquellos que son los más importantes para el usuario y aquellos que cubran las funcionalidades significativas).
Y como representación de la fase de elaboración se obtiene los artefactos que como arquitecto se deben desarrollar:

- Modelo de Análisis.
- Modelo de Diseño.
- Modelo de Despliegue.
- Modelo de Implementación.
- Documentación de la Arquitectura.

Modelo de Análisis: contiene clases del análisis y sus objetos organizados en paquetes que colaboran. En el modelo de análisis se tienen que identificar las clases que describen la realización de los CU, los atributos y las relaciones entre ellas. Con esta información se construye el Diagrama de clases del análisis, que por lo general se descompone para agrupar las clases en paquetes. Esta descomposición tiene impacto por lo general en el diseño e implementación de la solución.

Modelo de Diseño: es un modelo de objetos que describe la realización física de los casos de uso centrándose en como los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Sirve de abstracción de la implementación y es utilizada como entrada fundamental de las actividades de implementación.

Modelo de Despliegue: es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. El modelo de despliegue se utiliza

como entrada fundamental en las actividades de diseño e implementación debido a que la distribución del sistema tiene una influencia principal en su diseño.

Modelo de Implementación: Describe como los elementos del modelo de diseño, como las clases, se implementan en términos de componentes, ficheros de código fuente, ejecutables etc. El modelo de implementación describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y como dependen de los componentes unos de otros. [7]

Documentación de la Arquitectura: Aquí el arquitecto hace una descripción de la arquitectura donde recoge los diferentes aspectos necesarios y suficientes para la construcción de un Sistema.

1.7. Conclusiones

En la primera parte de este capítulo se hizo una pequeña introducción al tema de arquitectura, luego se realiza una breve reseña histórica, se explica porque es necesario la realización de una arquitectura. En una segunda parte, se realiza el estudio de algunos de los estilos y patrones candidatas a ser empleadas durante el desarrollo del software, sus características, ventajas y desventajas. Como resultado del análisis hecho, se pudo escoger las herramientas a utilizar durante el ciclo de vida completo del software, la decisión estuvo avalada por la política de uso de herramientas con soporte multiplataforma y licencias de utilización libre. Además el tema de la arquitectura como ha sido planteada en las bibliografías estudiadas y en las diferentes investigaciones realizadas, no es común que los proyectos desde hace unos cuantos años esté documentada, pero por la necesidad de la organización de desarrollo, la reutilización de diseño y códigos, y el mejor entendimiento del software se hace necesario la realización de una Arquitectura a la hora de desarrollar un software.

CAPÍTULO 2: CARACTERÍSTICAS DE LA ARQUITECTURA DEL SISTEMA.

2.1. Introducción.

En este capítulo se hace un análisis sobre los aspectos más importante que desempeña el Rol del Arquitecto durante la realización del Sistema de los Tribunales Militares de Región, trazándose una línea base de desarrollo, permitiendo la descripción de la Arquitectura, y recoge además todas las características de una forma u otra que tendrá el Sistema.

2.2. Actividades que debe cumplir un arquitecto para la realización de la Arquitectura del Sistema.

Ante de detallar las diferentes características que debe tener el Sistema, es bueno definir las actividades que debe desempeña el arquitecto, ya que juega un importante papel a la hora de elaborar o desarrollar un software.

El rol *Arquitecto de Software* conduce y coordina las actividades y los artefactos técnicos a través del proyecto. El Arquitecto de Software establece la estructura total para cada visión arquitectónica: la descomposición de la vista, la agrupación de elementos, y las interfaces entre agrupaciones mayores. Por lo tanto, en contraste con otros roles, la visión del *Arquitecto de Software* es más amplia en comparación con otras.

Las actividades principales que debe realizar el arquitecto son:

- Priorizar los Casos de Usos.

Definir los casos de usos como: críticos, secundarios, auxiliares u opcionales, lo que permite definir los módulos, subsistemas y escenarios así como la interacción entre ellos, que permite tomar decisiones hacia donde centrar los esfuerzos.

- Realizar el análisis arquitectónico.

Definir la arquitectura candidata del Sistema teniendo en cuenta, arquitecturas similares u otros sistemas, definir además los estilos arquitectónicos y patrones de la Arquitectura.

- Identificar los mecanismos de diseño.

Refinar el análisis de la arquitectura de acuerdo a las restricciones impuestas por el entorno de Implementación.

- Estructurar el Modelo de Implementación.

Facilita establecer la estructura en la que va estar la implementación del Sistema.

- Incorporar los elementos del diseño existentes.

Permite analizar las interacciones de las clases del análisis para encontrar interfaces, clases del diseño y subsistemas. Refinar la arquitectura e incorporar elementos reutilizables donde es posible.

Identifica las soluciones comunes a los problemas del diseño normalmente encontrados, incluye los elementos más significativos en la Arquitectura de Software.

- Identificar elementos del diseño.

Analizar las interacciones entre las clases de análisis para así identificar los elementos de diseños.

- Describir la Arquitectura en tiempo de ejecución.

Analizar requisitos de concurrencia, identificar los procesos, los mecanismos de comunicación entre dichos procesos e identificar el ciclo de vida de los procesos.

2.3. Descripción General de la Arquitectura del Sistema.

Según RUP, el arquitecto aparece durante todas las fase de desarrollo, pero solo juega un papel fundamental durante la fase de elaboración.

Al final de la fase de elaboración se ha desarrollado junto con los analistas del sistema los modelos del sistema que representan los CU, y sus requerimientos funcionales y no funcionales, además del análisis desde las perspectivas de la arquitectura. También se ha decidido que la arquitectura no sólo se ve condicionada por lo CU arquitectónicamente más significativos, como en muchas bibliografías, en especial la de Jacobson, Booch, Rumbaugh; es planteado, que no es suficiente, si no que además que se condiciona por los factores ya mencionados en capítulo anterior en la sección 1.2.3, como son:

2.3.1. Estilos Arquitectónicos que se van utilizar en el Sistema.

En la solución del software la identificación de un tipo de arquitectura es fundamental ya que facilitará la elaboración de un plano técnico para el diseño de la aplicación, que en este caso es una aplicación que brinda servicios a los clientes. Dentro de los estilos que se van utilizar en el sistema se encuentran:

- El Estilo Basado en Componentes: para la realización del diseño, basándose en la arquitectura candidata, que será mencionada posteriormente, se irán utilizando diferentes componentes, en los distintos niveles, como se muestra en la figura 2.1. Además en la realización de los diferentes modelos o diagramas que sirven para la representación y descripción de la arquitectura se utilizarán elementos de diseño y componentes para la implementación.
- El Estilo Orientado a Objetos: para la realización de la aplicación, siendo la misma una aplicación Web, fue seleccionado para su desarrollo un lenguaje PHP con su *Versión* 5, mencionado más adelante en la sección 2.2.5. Este es orientado a objetos por lo que se pueden diseñar las clases usando dicho paradigma de programación, ya que usarán manejadores de objetos (Object Handles), que son una especie de punteros que referencia los espacios en memoria donde residen los objetos. Cuando se asigna un manejador de objetos o se pasa como parámetro en una función, se duplica el propio object handle y no el objeto en sí, además para diseñar las diferentes clases de la aplicación Web orientadas a objetos se utilizan estereotipos, que son extensiones de UML, que dicha extensión son usada para el diseño de las clases mencionadas anteriormente.
- El Estilo Model-View-Controller (MVC) será utilizado como un patrón de diseño, explicándose en la subsección siguiente, patrones.
- El Estilo Orientado a Servicios: la seguridad del Sistema constituye un servicio externo, producto de un Sistema de Gestión de Seguridad realizado por desarrolladores pertenecientes al proyecto UCI_MINFAR, con el objetivo de estandarizar los componentes de seguridad dirigidos a los diferentes proyectos. El mismo brinda ayuda a los clientes en todos los aspectos relacionados con la planificación de la seguridad, incluyendo la gestión de la identidad de los usuarios, el control de accesos y el desarrollo de políticas consistentes de gestión de la seguridad.

Estos estilos se estarán utilizando de forma combinada según sus ventajas y desempeño arquitectónico debido a que la aplicación del Sistema de los Tribunales Militares se fundamenta en un diseño modular

gestionado a través de capas, que permiten el desarrollo incremental del Sistema, es decir, la arquitectura a utilizar sería una arquitectura en capas.

- El Estilo Basado en Capas: Mediante este modelo de diseño se facilita el desarrollo del Sistema debido a que se puede llevar a cabo en varios niveles y en caso de algún cambio sólo se ataca al nivel requerido sin tener que revisar entre código mezclado.

Representación Arquitectónica del Estilo Basado en Capas:

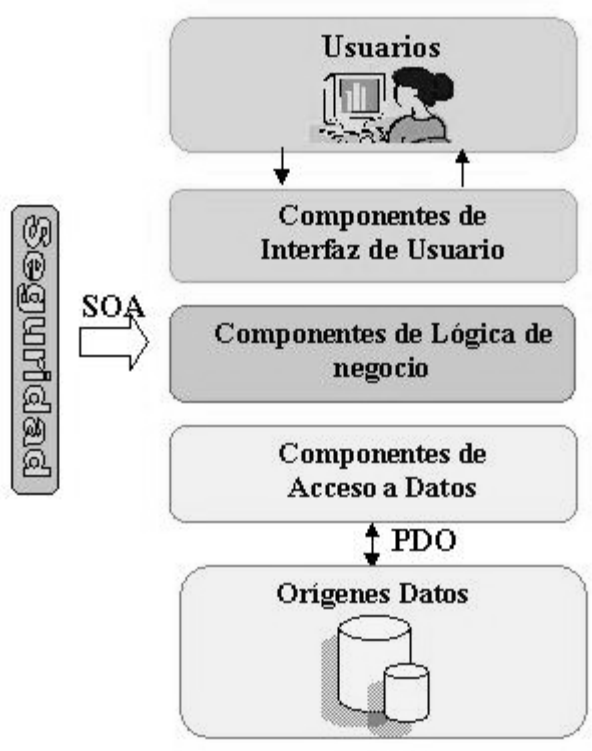


Figura 2.1. Arquitectura en capas. Tipos de componentes utilizados.

La figura muestra los tipos de componentes de software más comunes utilizados para la solución arquitectónica propuesta para Sistema con el tipo de arquitectura en capa, que se describe a continuación.

Los tipos de *componentes* identificados en el escenario de diseño son:

1. Componentes de Interfaz de Usuario. La mayor parte de la solución necesita ofrecer al usuario un modo de interactuar con la aplicación de Sistemas de Tribunales Militares de Región, que se está desarrollando. Las interfaces de usuario se implementan utilizando formularios, controles u otro tipo de tecnología que permita procesar y dar formato a los datos de los usuarios, así como adquirir y validar los datos entrantes procedentes de estos.

2. Componentes de lógica de negocio. Independientemente de si el proceso consta de un único paso o de un flujo de trabajo organizado, la aplicación requiere el uso de componentes que implementen reglas y realicen tareas.

3. Componentes de acceso a datos. La aplicación y los servicios necesitan obtener acceso a un almacén de datos en un momento determinado del proceso judicial. Por ejemplo, la aplicación de los Tribunales Militares de Región necesita recuperar los datos de un acusado de la *base de datos* para mostrar al usuario los detalles de los mismos, así como insertar dicha información cuando un usuario realiza una revisión. Por tanto, es razonable abstraer la lógica necesaria para obtener acceso a los datos en una capa independiente de componentes lógicos de acceso a datos, ya que de este modo se centraliza la funcionalidad de acceso a datos y se facilita la configuración y el mantenimiento de la misma.

4. Componentes de seguridad**:** Se utilizará como un servicio ya que los procesos judiciales requieren el uso de la funcionalidad proporcionada por un servicio externo proporcionando la directiva de seguridad que se ocupa de la autenticación, autorización, comunicación segura, auditoria y administración de perfiles.

2.3.2. Descripción de los patrones que se van a utilizar en el Sistema.

Para describir los diferentes patrones que el Sistema utilizará, según el tipo de arquitectura en la cual se fundamenta el diseño, se analizará el o los patrones, de acuerdo a la capa donde interactúan. A continuación se abordarán dichos patrones que serán empleados en dependencia del entorno en el que se trabaje.

Se utilizarán patrones de diseño, que proporcionan un esquema para refinar los subsistemas o componentes de un sistema software y las relaciones entre ellos. Describen estructuras recurrentes para comunicar componentes que resuelven un problema de diseño en un contexto particular. Son patrones de un nivel de abstracción menor que los patrones de arquitectura. Están por lo tanto más próximos a lo que sería el código fuente final. Su uso no se refleja en la estructura global del sistema. En la actualidad son muchos los patrones de diseños utilizados en la construcción de una aplicación, como se hace referencia en el capitulo anterior.

- **Capa de Presentación.**

Para la capa de presentación se utilizarán los siguientes patrones:

El patrón Observer, que está dentro de los patrones del GoF, que proporciona a un objeto un mecanismo para avisar a otros objetos de cambios en el estado de la aplicación. En el caso de una aplicación Web, se puede utilizar para que la aplicación reaccione a los eventos generados por el usuario, ejecutando en reacción a ellos la lógica que se ha definido.

El patrón Model-View-Controller (MVC)**,** con este patrón se consigue separación de responsabilidades. Se trata de dividir una aplicación o una tarea en la vista (la interfaz), el controlador (lo que maneja la vista), y el modelo (lo que utiliza el controlador para generar nuevas vistas). Esta descripción está hecha a grosso modo. Pero ahora en este caso se habla de patrones dentro de la capa de presentación, por lo que alguno puede dudar sobre por qué se incluye en este listado. Pues muy sencillo: en esta aplicación global (o sea, la aplicación entera) se tienen ciertas tareas que deben ir en la capa de negocio (recordar que es la que se encarga de toda la lógica de negocio de la aplicación), pero además se tienen otras tareas que se encargan de la lógica de presentación (la que se encarga de gestionar el estado y comportamiento de la interfaz), que, aunque es lógica también, no debe ir en la capa de negocio. Pues es aquí donde se aplican

el MVC de nuevo: dentro de la capa de presentación se separan por un lado las vistas (páginas Web) del controlador (los controladores que se comentan anteriormente), y se toman como "modelo" el resto de aplicación, es decir, la capa de negocio y la de acceso a datos.

El patrón Application Controller**:** este patrón se encarga de gestionar la interacción entre el usuario y la aplicación, dirigiendo el flujo de navegación y controlando el estado de la sesión. Se puede utilizar si se quiere que la navegación no sea lineal, sino que pueda variar, por ejemplo: "si el usuario pincha aquí y es un usuario normal, que vaya a ésta página, pero si pincha un usuario administrador, que vaya a esta otra". Además de ofrecernos esta característica, nos ofrece control sobre el estado de la sesión del usuario.
El patrón Facade /Fachada: que proporciona una interfaz unificada para un conjunto de de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar. Como es el caso de las interfaces de los tres subsistema diseñados en el sistema.

- **Capa de Lógica de Negocios.**

Dentro de las características que debe cumplir el diseño de un sistema, es importante resaltar los aspectos arquitectónicos que permiten un mejor funcionamiento, por tanto se hace necesario la utilización los patrones GRASP (General Responsability Asignment Software Paterns, Patrones Generales de Software para Asignar Responsabilidades), dentro de estos se utilizarán:
El patrón Bajo Acoplamiento, el mismo resolvería el problema de dar soporte a una dependencia escasa y a un aumento de la reutilización, ya que esto consiste en que el acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, una clases con bajo acoplamiento no depende de muchas otras clases. Mientras que una clase con alto acoplamiento recurre a muchas otras clases, este tipo de clases no es conveniente ya que presenta los siguientes problemas:

- Los cambios de las clases afines ocasionan cambios locales.
- Son más difíciles de entender cuando están aisladas.
- Son más difíciles de reutilizar porque se requiere la presencia de otras clases de las que dependen.

El patrón de Alta Cohesión, mantiene la complejidad dentro de los límites manejables, caracteriza a las clases con responsabilidades estrechamente relacionadas con un trabajo enorme. Ahora bien, el trabajo

en un sistema que contenga clases con una baja cohesión hace el trabajo excesivo, ya que las clases son más difíciles de comprender, reutilizar, conservar y son muy delicadas: las afectan constantemente los cambios.

Estos dos patrones son un principio que debe cumplir siempre el sistema y se deben tener presentes en todas las decisiones de diseño. También permiten hacer una evaluación del comportamiento de los elementos de un componente y sus colaboraciones.

- **Capa de Acceso a Datos.**

En el sistema se usaran los conceptos de Factory (Factoría) y Abstract Factory (Factoría Abstracta).El patrón factoría es uno de los varios patrones creadores definidos por la GoF. La idea que se esconde detrás de este patrón es la de centralizar el sitio donde se crean los objetos, normalmente donde se crean objetos de una misma "familia", sin dar una definición clara de lo que el software puede entender como familia, como podría ser componentes visuales, componentes de la lógica del negocio, o objetos concurrentes en el tiempo.

La clase factoría devuelve una instancia de un objeto según los datos que se le pasan como parámetros. Para que la creación centralizada de objetos sea lo más "útil y eficaz" posible, es de esperar que todos los objetos creados desciendan de la misma clase o implementen la misma interfaz (es decir, hagan una operación similar pero de distintas formas), así se podrán usar todos de la misma manera, con los mismos métodos (gracias al *polimorfismo*), sin importar que clase concreta se está tratando en cada momento.

El patrón Factoría Abstracta es muy sencillo si se ha entendido el patrón factoría. Como la palabra abstracta se puede suponer, este patrón lleva al de la factoría un punto más lejos en la idea de abstraer el código de creación de objetos del resto de la aplicación. ¿Cómo debe entender esto?, pues una factoría abstracta es una clase factoría, pero que los objetos que devuelve son a su vez factorías. Por este motivo, para que sea efectiva, estas factorías que devuelve, deben ser de la misma familia (es decir, tener antecesores comunes), como ocurría con las factorías normales.

El patrón Table Data Gateway consiste en crear una instancia por cada tabla existente en la BD. Sus métodos consisten en las operaciones básicas que se realizan sobre estas tablas, insertar, modificar y eliminar.

Además de estos patrones de diseño y de guiarnos por las estrategias antes explicadas en el capítulo anterior, subsección 1.4.1, se pueden encontrar los patrones de la arquitectura, como:

- Client/Server, Three-Tier, Pee-to-Peer, que sirven para comprender el hardware del sistema que se construye, y ayuda a diseñar el mismo, ya que Client/Server se pone de manifiesto de la siguiente forma, la distribución del mismo es, que tiene un nodo cliente que ejecuta el código de la interfaces de usuario y parte de la lógica de negocio (clase de control) en cada puesto de trabajo (PC). Los nodos servidores mantienen las funcionalidades del sistema permitiendo verificar los diferentes registros. El patrón Three-Tier es una arquitectura cliente/servidor donde la interfaz del usuario, la lógica del proceso funcional y el acceso de los datos, mantienen separadas las plataformas como módulos independientes. El patrón Pee-to-Peer que se traduciría par a par o de punto a punto que se refiere a una red que no tiene Cliente ni Servidores fijos, sino una serie de nodos que se comportan simultáneamente como clientes y como servidores de los demás nodos de la red, permitiendo la conexión. Estos patrones definen una estructura para el modelo de despliegue y surgieren como se deben asignar los componentes a los nodos.
- El patrón Layers define como organizar el modelo de diseño en capas, lo cual quiere decir que los componentes de una capa sólo puede hacer referencia a componentes en capas inmediatamente inferiores. Este patrón es importante porque simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo las dependencias de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores. Además ayuda identificar que reutilizarse, proporciona una estructura que ayuda a tomar decisiones sobre que parte se va a construir.

2.3.3. Marco de trabajo del Sistema.

Marco de Trabajo o Framework es una estructura de soporte definida donde el software puede ser organizado y desarrollado. Típicamente, un marco de trabajo puede incluir soporte de programas, librerías y un lenguaje de scripting entre otros software para ayudar a desarrollar y unir los diferentes componentes del mismo.

El marco de trabajo representa una Arquitectura de Software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

Para definir el marco de trabajo, proporcionando una filosofía de trabajo, librerías y funciones para hacer más fácil el desarrollado del Sistema, se propone el siguiente donde el mismo va estar en correspondencia con el estilo de aplicación.

El marco de trabajo y los estilos de aplicaciones se forma con los componentes de interfaz de usuario, los componentes de lógica de negocio y acceso a datos principalmente.

En la figura 2.2 se muestra la distribución de los componentes en las diferentes capas y el papel que juegan las clases desarrolladas para formar el marco de trabajo del sistema. El ejemplo se basa en un caso de actualización para un proceso simple como es el de este sistema.

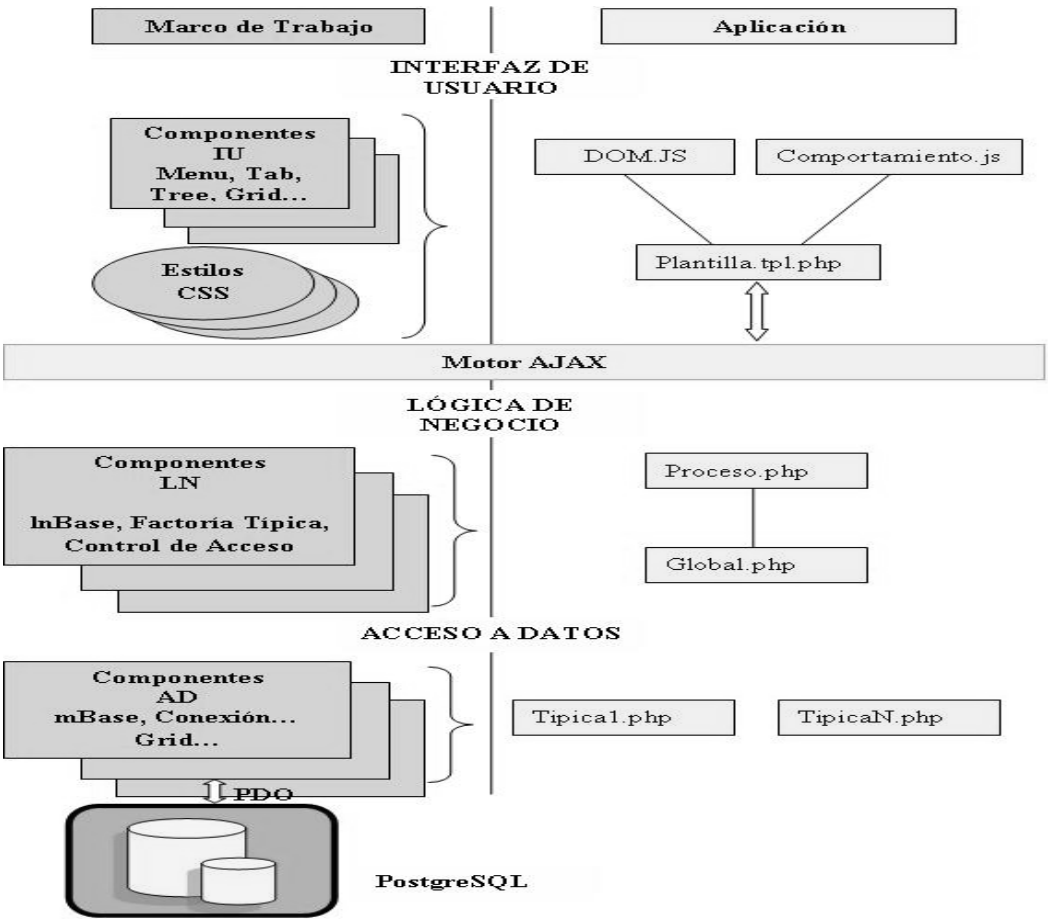


Figura 2.2. Marco de Trabajo General

Distribución física de los componentes del marco de trabajo.

En la presente sección se pretende describir como distribuir físicamente los componentes del marco de trabajo en la aplicación.

Figura 2.3. Distribución física del Marco de Trabajo.

Descripción del Marco de Trabajo.

El Marco de Trabajo va estar estructurado en tres capas, ya que es el estilo a utilizar dando la posibilidad de dividir en niveles físicos:

En la primera capa, se encuentra la capa de presentación donde están los diferentes componentes y estilos, siendo estos java script, imágenes y estilos css.

En la segunda capa, capa de lógica de negocio se encuentran las diferentes clases de lógica de negocio.

Y en la capa de acceso a datos, se van encontrar las configuraciones de las conexiones hacia la base de datos y las relaciones dinámicas de los diferentes reportes.

2.3.4. Estándares y políticas corporativos adaptadas al Sistema.

Para obtener un producto de Software que resulte del agrado del usuario se definieron estándares a seguir para la construcción del Sistema, siendo unos de los factores por lo que se condiciona la arquitectura.

Se encuentran divididos de la siguiente manera:

- **Estándares de Diseño.**

La página principal de la aplicación, se concibe como un portal, que permite la autentificación del sistema; con un menú, que tiene todos los procesos que se realizarán durantes la planificación y ejecución de un Juicio Oral, este no debe exceder de 3 niveles de profundidad, además es donde se agrupan las funcionalidades del sistema.

Las páginas deben tener una cabecera (banner) representativa, un área de trabajo, con los formularios correspondientes y una barra menú con las opciones.

Se trabaja con una hoja de estilo en común para lograr la uniformidad, donde se utiliza la familia de fuentes Tahoma, el tamaño de la misma no debe diferir mucho de 11 a 14 Px.

Los colores con los que se trabajarán serán tonalidades claras y fuertes basadas en el azul, verde combinados con el color blanco y gris.

- **Estándares de Codificación.**

Se han realizado una descripción de los estándares de codificación a seguir para implementar el Sistema a partir de los que se han definidos por el cliente para diseños Web, lenguaje PHP y Gestor PostgreSQL, y para obtener esta información se hace necesario ver el Anexo 1.

- **Estándares de Organización.**

Estos estándares surgen para darle organización a la hora de la elaboración del Sistema. Ya que se definen la estructura de las carpetas en el sitio, los nombres de los formularios, entre otros. Se tiene estructurado de la siguiente forma:

- Carpetas del sitio en minúsculas y español, y tendrá la siguiente estructura:
    - Clases.

- Estilos.
- Js.
- Nomencladores.

- Procesos y dentro de esta la carpeta de plantillas
  - Imágenes.

Cuidar de no tener ficheros con muchos subniveles en el sitio, tratar de tener todos los directorios en la raíz del sitio.
Mantener en *JavaScript* los mismos estándares de código que en PHP.

2.3.5. Requisitos no funcionales.

En esta sección se describe los principales requisitos que estarán presentes en el Sistema y que tienen impacto en la Arquitectura como son la seguridad, portabilidad, reusabilidad, etc, entre ellos se pueden encontrar:

Apariencia o interfaz externa:

- La interfaz a implementar debe ser sencilla, para que los usuarios que no son personas expertas en la rama de la informática, no necesiten tanto tiempo de adiestramiento.
- Por el uso diario y constante que tendrá el software, la interfaz debe ser agradable, que favorezca el estado de ánimo del cliente y que combine correctamente los colores, tipo de letra y tamaño y que los iconos estén en correspondencia con lo que representan.
- Deben utilizarse plantillas con un mismo estilo.

Usabilidad:

- El sistema debe ser de fácil manejo para los usuarios que tengan niveles básicos sobre la computación o hallan trabajado con la Web.
- Debe tener una opción de ayuda sobre las principales operaciones que se realizan y sus iconos respectivos para lograr un menor tiempo de aprendizaje.

- El sistema simulará tal y como es el proceso judicial para lograr el menor tiempo en cuanto a la comprensión del sistema y del proceso.

Rendimiento:

- La aplicación debe estar concebida para el consumo mínimo de recursos.
- El sistema debe ser capaz de formular la respuesta lo más rápido posible.

Soporte:

Para el servidor de aplicaciones:

- Se requiere que esté instalado un intérprete de ficheros PHP y con las últimas actualizaciones del lenguaje.

Para el servidor de base de datos:

- Se requiere que esté instalado un gestor de base de datos que soporte grandes volúmenes de datos y velocidad de procesamiento.

Para el cliente:

- Se requiere esté instalado un navegador que interprete *Javascript* y versiones HTML 3.0 o superior.

Portabilidad:

- El sistema deberá ser compatible con el sistema operativo UNIX (Linux).
- El sistema deberá ser compatible con el sistema operativo Windows (Versiones como 2000 y XP), siendo además accesible principalmente en la Intranet con el navegador Mozilla.

Hardware:

Para las computadoras del cliente:

- Se requiere tengan tarjeta de red.
- Se requiere tengan al menos 64 MB de memoria RAM.
- Se requiere al menos 100MB de disco duro.
- Procesador 512 MHz como mínimo.

Para los servidores:

- Se requiere tarjeta de red.
- Se requiere tenga la menos 256MB de RAM.
- Se requiere al menos 1GB de disco duro.
- Procesador 1.2 GHz como mínimo.

Dispositivo: Impresora

- Velocidad de impresión: Calidad casi carta 77 cps (10 cpp).

Software:

- El sistema se desarrollará con tecnología PHP *versión* 5.0 o superior.
- Se utilizará un servidor con el sistema operativo instalado Windows 2000 o superior, o con sistema operativo UNIX (Linux) preferentemente.
- Se utilizará tecnología Apache *versión* 2.0 o superior para el servidor Web.
- El sistema utilizará una base datos implementada en PostgreSQL *versión* 8.0.
- En las computadoras de los clientes se garantizará versiones de Windows 2000 o superior, así como Linux y sus correspondientes distribuciones.
- En las computadoras de los clientes solo se requiere de un navegador (Internet Explorer *versión* 4.0 o superior, Mozilla Firefox *versión* 1.5 o superior.
- En caso de que el usuario no contara con los recursos suficientes para que la aplicación funcione con la arquitectura descrita entonces la computadora tiene que tener instalados todos los programas antes mencionados.

Seguridad:

- El sistema debe comunicarse usando un protocolo seguro (*http*s).
- Los datos no pueden viajar de forma transparente por la red, deben ser encriptados.
- Chequear si el usuario que está accediendo al sistema está autenticado y brindarle servicio de autenticación.
- Mantener la integridad de la información, es decir que no se pierda durante su almacenamiento o transporte.
- Permitir que cuando se borre cualquier documento o información pueda existir una opción de advertencia antes realizar la acción.

- Realizar auditoria a los principales eventos dentro del sistema, registrando al usuario, el tipo de usuario y los eventos efectuados.

Confiabilidad:

- La información manejada por el sistema está protegida de acceso no autorizado y divulgación.

Integridad:

- La información manejada por el sistema será objeto de cuidadosa protección contra la corrupción.

Fiabilidad:

- Debe garantizarse el resguardo de la información, de modo que estén duplicados, así como la grabación periódica de la Base de Datos, de forma tal que se posibilite la reinstalación del sistema y los datos, en caso de algún problema presentado en la explotación del mismo.

Legales:

- El sistema se basa en el manual de normas y principios establecidos por el MINFAR.
- La mayoría de las herramientas de desarrollo son libres y del resto, las licencias están avaladas.
- El sistema tendrá en cuenta lo establecido por la ‘’Ley de procedimiento penal Militar’’ y en la ‘’Ley número 97, de los Tribunales Militares’’, en todo lo referido al desarrollo del trabajo judicial que se lleve al nuevo Sistema.

2.3.6. Distribución especifica del sistema.

El sistema interactúa con el usuario mediante una interfaz Web muy fácil de utilizar. Su diseño es sencillo, con pocas entradas, permitiendo que no sea necesario mucho entrenamiento para utilizar el sistema. El software podrá ser usado bajo los sistemas operativos Windows (versiones como 2000 y XP), y Linux. Se debe disponer en el servidor con el sistema operativo instalado Windows 2000 o superior, o con sistema

operativo UNIX (Linux) preferentemente. Se utilizará como lenguaje de programación: PHP *versión* 5.0 y como gestor de Base de Datos: PostgreSQL 8.0.

Navegador compatible o superior con Internet Explorer *versión* 4.0 o superior, Mozilla Firefox *versión* 1.5 o superior con las extensiones Web Developer, FireBug e InspectThis. Debe garantizarse una protección contra acciones no autorizadas o que puedan afectar la integridad de los datos. Por razones de seguridad se desea conservar el servidor de *base de datos* separado del servidor de aplicaciones.

2.4. Línea base de la Arquitectura en el Sistema.

Después determinar los factores que dan la posibilidad de forjar la arquitectura de una forma adecuada, organizativa y un análisis profundo; más la información adquirida en la primera fase, realizada por el analista del sistema, el arquitecto obtiene una primera *versión* de lo que es la línea base de la arquitectura. La línea base de la arquitectura es una *versión* interna del sistema, que está centrada en la descripción de la arquitectura, y serviría como la estrategia de trabajo a seguir.

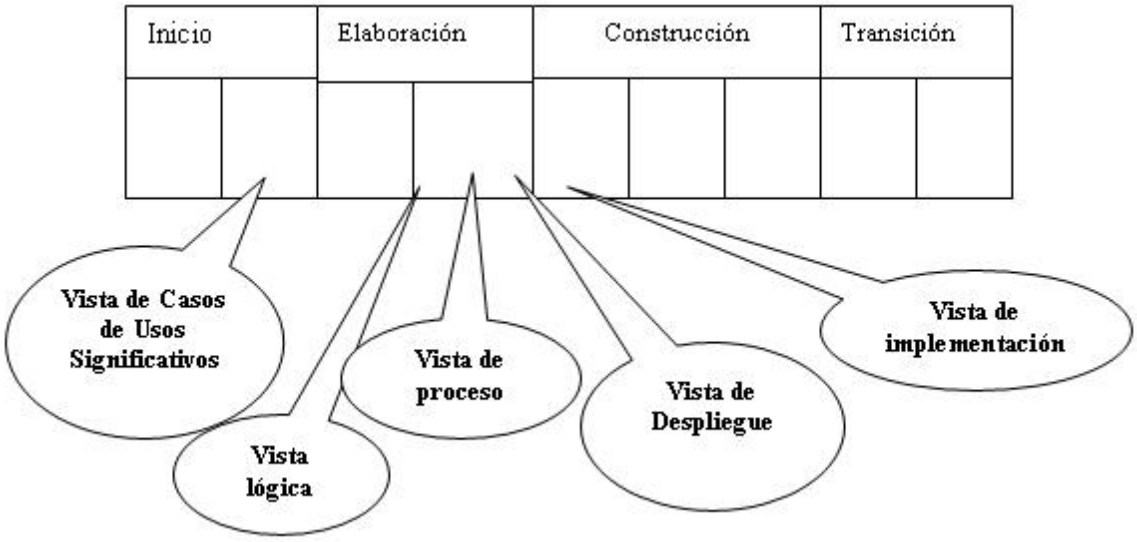


Figura 2.4. Línea base de la Arquitectura y las versiones internas del Sistema.

Descripción de la línea base:

Según RUP, cada ciclo de trabajo produce una *versión* del Sistema, que cada *versión* es un producto que incluye los artefactos que se crean durante la vida del proyecto, como en este caso, la obtención de los diferentes modelos que permitirán la representación de la descripción de la arquitectura.

La descripción de la arquitectura se obtiene de dichas versiones, además es un extracto o un conjunto de vistas de los modelos que están en la línea base de arquitectura. A continuación se explicará como se representa la arquitectura.

¿Cómo se representaría la arquitectura del Sistema?

Se representa a través de 4 + 1 vista, mencionada en el capitulo 1; dicho de modo que se entienda, 4 de estas vistas están regidas por una rectora. La rectora se considera la vista de casos de uso, y las restantes son la vista lógica, la vista de procesos, la vista de despliegue y la vista de implementación.

Vista de casos de uso: Esta vista representa un subconjunto del artefacto Modelo de casos de uso y lista los *casos de usos* o escenarios del modelo de casos de uso más significativos, con las funcionalidades centrales del sistema. Si el sistema se hace extenso entonces se debería organizarse en paquetes, lo cual facilitaría la comprensión de la vista de casos de uso.

Aquí se representa el diagrama de casos de uso de los mismos con una breve descripción de cada uno.

Vista lógica: Esta vista representa un subconjunto del artefacto Modelo de diseño, la cual representas los elementos de diseño más importantes para la arquitectura del sistema. Este describe las clases más importantes, su organización en paquetes y subsistemas. También describe las realizaciones de casos de uso más importantes como por ejemplo las que describen aspectos dinámicos del sistema.

Vista de procesos: Esta vista suministra una base para la comprensión de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos concurrentes o hilos. En este caso, la aplicación de Tribunales Militares no realiza accesos concurrentes, por lo que la misma no será representada.

Vista de despliegue: Esta vista suministra una base para la comprensión de la distribución física de un sistema a través de nodos. Suele utilizarse cuando el sistema está distribuido. Y hay una traza directa del modelo de implementación, puesto que cada componente físico debe estar almacenado en un nodo, esto incluye también la asignación de tareas provenientes de la vista de procesos en los nodos.

Vista de implementación: Esta vista describe la descomposición del software en capas y subsistemas de implementación. También provee una vista de la traceabilidad de los elementos de diseño de la vista lógica ahora para la implementación.

2.5. Conclusiones

Se conforma la arquitectura a utilizar en el Sistema con la conjugación de los diferentes estilos arquitectónicos y patrones, los mismos han permitiendo, la organización y diseño a la hora de estructurar la arquitectura. La línea base de la arquitectura jugó un papel muy importante en la descripción de la arquitectura, definido por la metodología RUP a lo largo del ciclo de desarrollo, siendo este representado por las 4+1 vistas arquitectónicas, facilitando que los desarrolladores que intervienen en el proceso se comuniquen con un lenguaje común.

CAPÍTULO 3: DISEÑO DE LA ARQUITECTURA

3.1. Introducción

En este capitulo se harán las consideraciones necesarias para la construcción o diseño de la solución propuesta, se describirá la Arquitectura que va usar el Sistema, así como las vistas donde serán representadas, mediante los diferentes modelos o artefactos necesarios que permiten dicha descripción.

3.2. Vista de Casos de Uso

En esta sección se muestran los diferentes Casos de Usos (CU) o escenarios que representan las funcionalidades principales que tiene el Sistema, o sea los CU significativos para la Arquitectura.

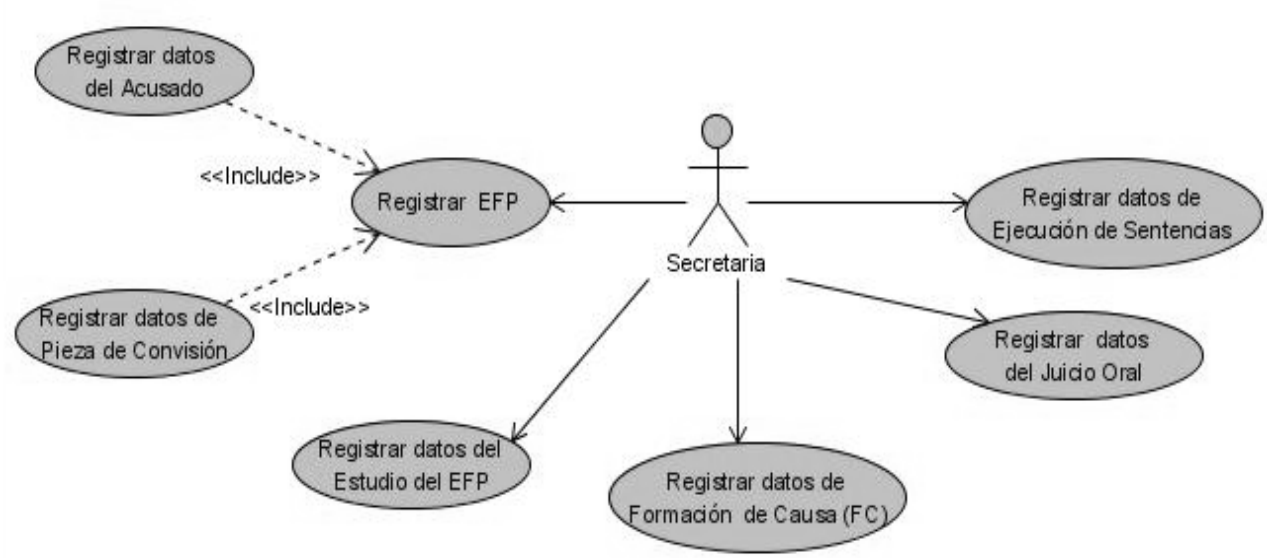


Figura 3.1. Vista de Casos de Usos arquitectónicamente significativos.

A continuación serán enumerados los mismos:

1. Registrar Expediente de Fase Preparatoria (EFP).
2. Registrar datos del Acusado.
3. Registrar datos de la Pieza de Convicción.
4. Registrar datos del Estudio del EFP.
5. Registrar datos de Formación de Causa (FC).
6. Registrar datos del Juicio Oral.
7. Registrar datos de Ejecución de Sentencia.

3.3. Vista Lógica

En esta sección se describe la parte más importante para la arquitectura del modelo de diseño, la descomposición de subsistema y/o paquetes y para cada paquete cuyo contenido sean significativos para la arquitectura, donde se presenta su composición en términos de clases y sus responsabilidades, y atributos principales.

3.3.1. Descripción Global.

La siguiente figura representa la división del Sistema en subsistemas, esto se hace con el objetivo de hacer que el mismo sea más re-usable y facilite el mantenimiento, ya que un subsistema es un agrupamiento semánticamente útil de clases o de otros subsistemas.

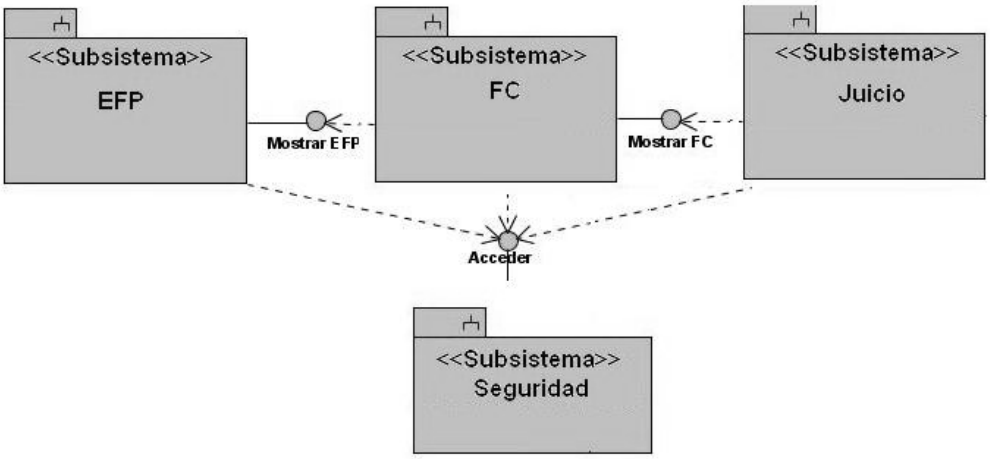


Figura 3.2. Vista de los Subsistemas.

Descripción de los subsistemas:

- EFP: Contiene las Realizaciones de los Casos de Usos correspondientes a la Gestión de EFP.
- FC**:** Contiene las Realizaciones de los Casos de Usos correspondientes a la Gestión de Formación de Causa.
- Juicio: Contiene las Realizaciones de los Casos de Usos correspondientes a la Gestión de Juicio Oral.
- Seguridad**:** Contiene las Realizaciones de los Casos de Usos correspondientes a la Seguridad.

3.3.2. Paquetes de diseño arquitectónicamente significativos.

En esta subsección describe la descomposición global o general en que se dividirá el modelo de diseño, en términos de paquetes, dependencias y capas. Incluye en ella una breve descripción y la representación de todas las clases que lo componen

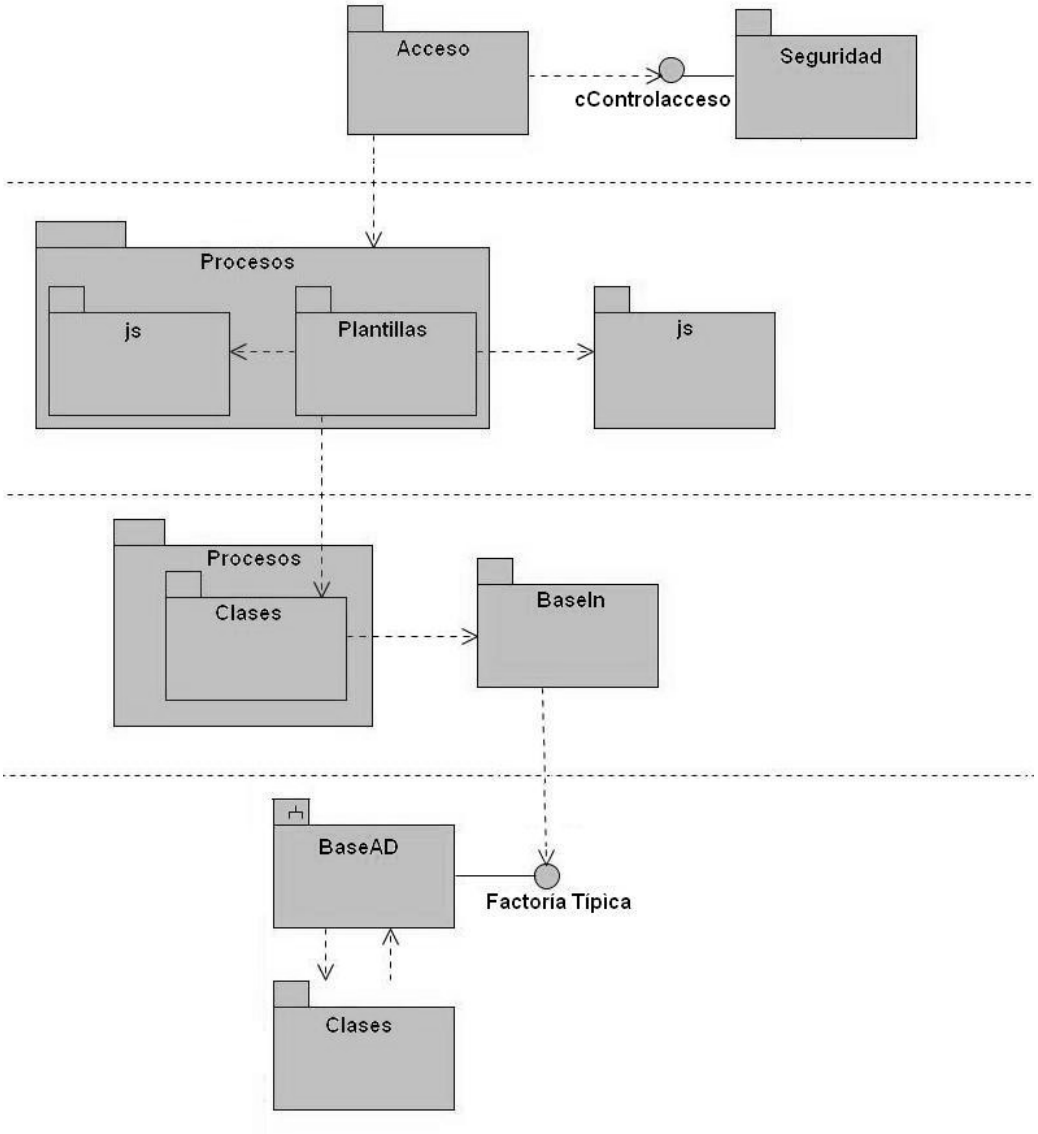


Figura 3.3 Vista de los Paquetes por Capa.

Descripción de los elementos de diseño de cada paquete por Capa.

- **Capa de Presentación.**

Se encuentra el paquete de Acceso que accede al paquete Seguridad que está desarrollado en otro sistema. Además de los paquetes Proceso que dentro tiene otros paquetes con los js de las plantillas, que también son un paquete, en la parte de afuera está el paquete js que contiene los DOM.

Descripción de los paquetes:

- ✓ Paquete de Seguridad.

Este es un sistema externo al de Tribunales Militares que está implementado por el grupo de desarrollo UCI-FAR que es usado por todos los Sistemas desarrollados por el mismo, el cual permite identificar al usuario actual dentro del grupo al que pertenece y los componentes a los que puede acceder a través del menú.

- ✓ Paquete de Acceso.

Index: Tiene la función de llamar a la página autentica.

Autentica: Esta llama a la página principal y permite el autenticación del usuario.

Redireccionar: Tiene la función de buscar las páginas según el proceso a realizar.

Server: Es el encargado de gestionar el acceso con la página servidora para acceder a la página cliente, según su rol.

- ✓ Paquete de Procesos.

Este contiene los paquetes js y plantillas.

- Paquetes js.

Está los js de cada CU del Sistema, y son los encargado de todas las acciones de la parte del cliente que van estar asociada a cualquier Pág. que se construya.

- Paquete plantillas.

Página cliente: Tiene como función de mostrar la información, incluye un formulario. En correspondencia con la página cliente están los JS de cada Caso de Uso, DOM, dhtmlx.

Formulario: Es el área de trabajo donde se introduce los datos de la información Referente al CU.

- ✓ Paquete js.

Este es el paquete que contiene.

DOM.js: Son los encargados de realizar acciones en funciones de java script como validar dato, cerrar marco, centrar ventana, cambiar apariencia, además usa validaciones AJAX que permiten que la respuesta en la parte del Cliente sea rápida.

Dhtmlx.js: Son ficheros js de los componentes dhtmlx control donde se pueden encontrar (Menú, Toolbar, Grip, Tabbar).

Además de estos paquetes existen los siguientes elementos de diseño que son los que permiten la comunicación o el acceso al paquete de Seguridad.

Página Servidora: Es la encargada de unir todos las funcionalidades entre la capa de presentación y la lógica de negocio. Que tiene incluido los siguientes elementos.

Global: Está compuesta por la función script_rutime que es la encargada para el calculo del tiempo de ejecución de un script y la función de autoload, es la función mágica que garantiza la carga automática de las clases al instanciar. Además incluye controlacceso.inc

Controlacceso.inc: Hace una instancia a cControlacceso, define la variable *http* que debe contener la dirección de la página del formulario de autenticación, calcula el URL de la página principal y verifica la tercera capa de seguridad, permitiendo que el usuario pueda ver la página a acceder.

CControlacceso: Tiene la función de control de acceso a la página, control de acceso a las funcionalidades, chequea que un certificado sea válido, también tiene algoritmos para conformar números de chequeos de páginas y funcionalidades, además tiene la función de autenticar usuario.

A continuación se hace la representación del Diagrama de Clases del Diseño con un ejemplo de uno de los CU del Sistema, pero solo corresponde a la Capa de presentación:

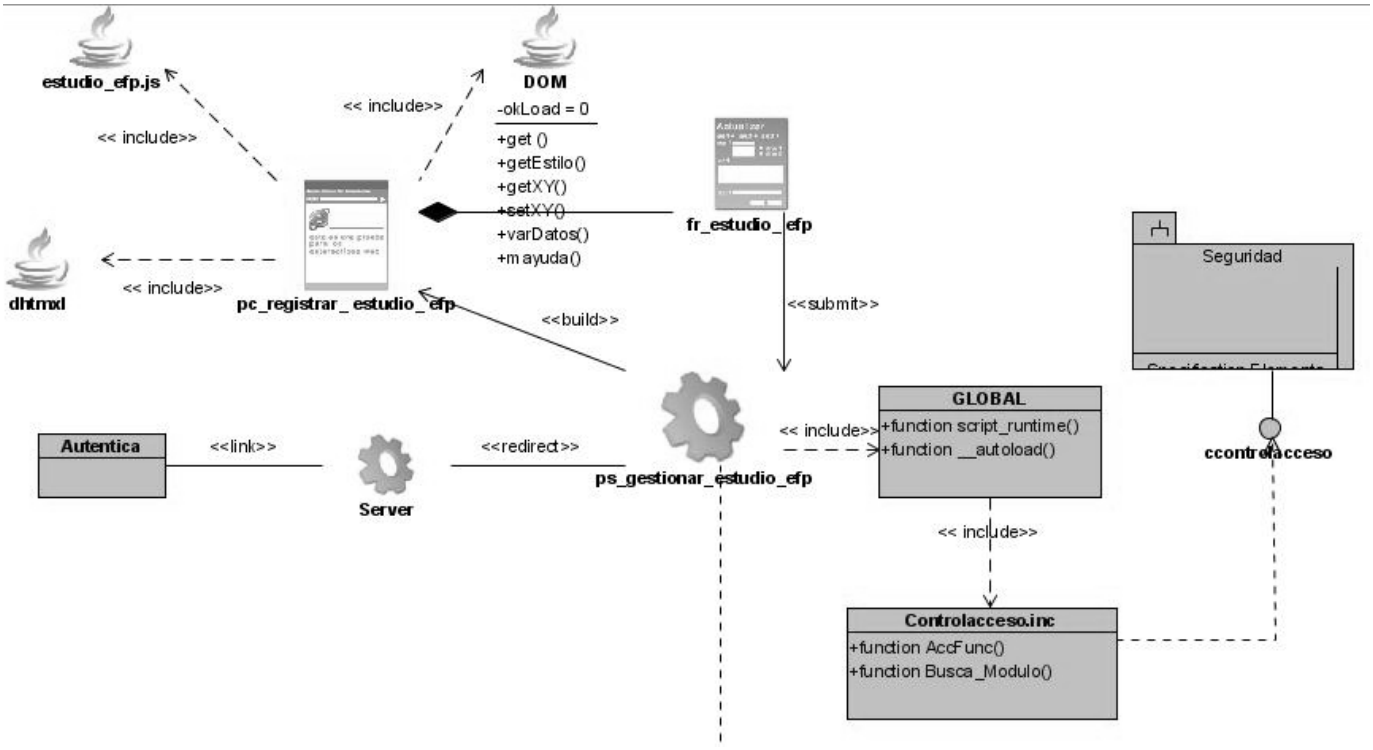


Figura 3.4. Vista de Diseño de la Capa de Presentación.

- **Capa Lógica de Negocio**

En la lógica de negocio se encuentra el paquete de Proceso que contiene a su vez el paquete de clases que depende del paquete Baseln

Descripción del los paquetes:

- ✓ Paquete de Proceso.

Aquí se encuentran los diferentes formularios, de forma programada con su diseño y los enlaces la lógica de negocio, estos formularios van a ser extensión php y van a coincidir con los diferentes procesos que se realizan a la hora de un procesamiento Judicial.

- ✓ Paquete clases.

Se encarga de hacer referencia a Baseln, donde están las diferentes clases de lógica de negocio lnCUX.

LnCUx: Lógica de negocio de uno de los CU del Sistema. Esta puede ser de dos tipos lnActualizar, que es la encargada de recoger todos los métodos que permiten actualizar los datos en la *base de datos*. Y la

lnReporte, que es la encargada de recoger todos los métodos que permiten visualizar al usuario la información más importante. La misma está relacionada con Miscelánea y grip, y esta hereda de lnbase.

Miscelánea: Encapsula métodos genérico de muchas clases que serán utilizadas.

Grip: Representa las clases de marco de trabajo para el empleo de los componentes dhtmlx (Menú, Toolbar, Grip, Tabbar, árbol).

Lnbase: Es la encargada de hacer instancia a Factoría Típica, además de almacenar la dirección de salida de los CU.

Representación del Diagrama de Clases del Diseño con un ejemplo de uno de los CU del Sistema, pero solo corresponde a la Capa de Lógica de Negocio.

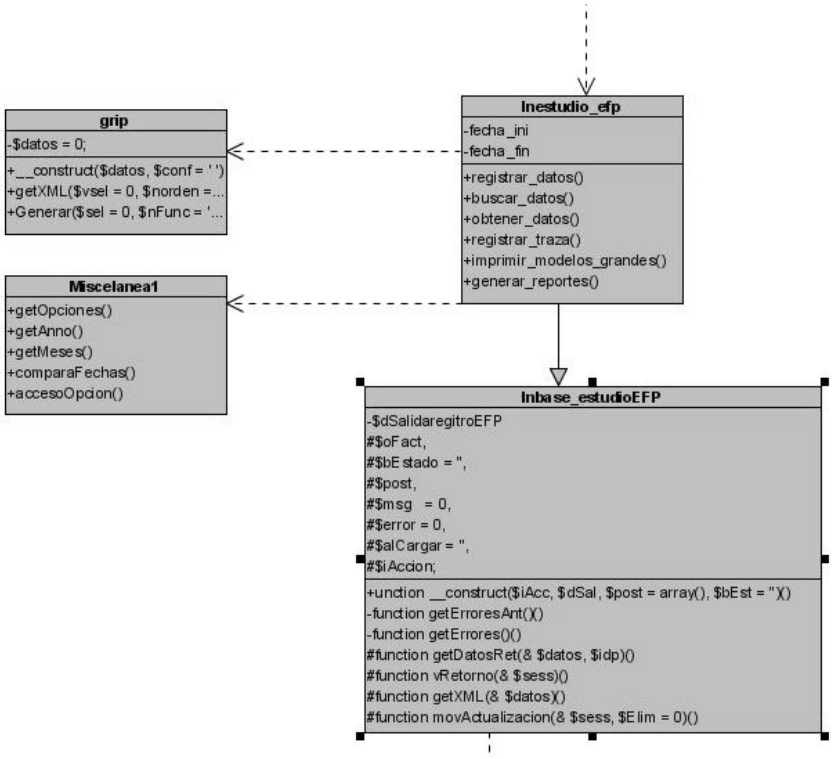


Figura 3.5. Vista de Diseño de la Capa de Lógica de Negocio.

- **Capa de Acceso a Datos.**

Se encuentran el paquete BaseAD que contiene una interfaz Factoría Típica brindándole información a la capa superior. Además este paquete intercambia información con el paquete de clases.

- ✓ El paquete AD.

Factoría Típica: Implementa métodos llamado call, que nos permiten de forma transparente la interacción con el resto de las clases del modelo de persistencias. Además implementa de forma explicita tres métodos base:

Crear -> Permite la creación de instancias de clases. Emplear solo cuando se desea ejecutar una única operación de las que implementa la clase instanciada (siempre se destruirá automáticamente la instancia después de la ejecución del método) y además su constructor debe recibir parámetros.

pCrear -> Permite la creación de instancias de clases. Emplear cuando se desea ejecutar varias operaciones de las que implementa la clase instanciada. La destrucción del objeto creado será responsabilidad del programador, aunque al concluir la ejecución del script esta se destruye.

Destruir -> Permite destruir una instancia creada.

También implementa de forma implícita el resto de las operaciones que están contenidas dentro de las clases del modelo de acceso a datos, las más empleadas son por ejemplo:

Insertar -> Para insertar datos en la entidad dada.

Modificar -> Para modificar datos en la entidad dada.

Eliminar -> Para eliminar datos en la entidad dada.

getDatos -> Para consultar datos en las entidades de nomenclaturas.

- ✓ Paquete de clases.

Este paquete contiene todas las consultas y típicas de los datos, y nomencladores de las diferentes formularios según los procesos.

A continuación se realiza la representación del Diagrama de Clases del Diseño con un ejemplo de uno de los CU del Sistema, pero solo corresponde a la Capa de Acceso a Datos.

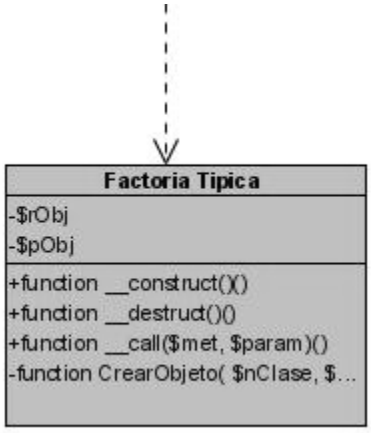


Figura 3.6 Vista de Diseño de la Capa de Acceso a Datos.

3.3.3. Realizaciones de un Caso de Uso

En esta sección se describe como funcionaría el sistema, dando dos ejemplos de cómo se llevarían a cabo en términos de clases, un CU de los que han sido considerado uno de los más importantes y dos secciones del mismo.

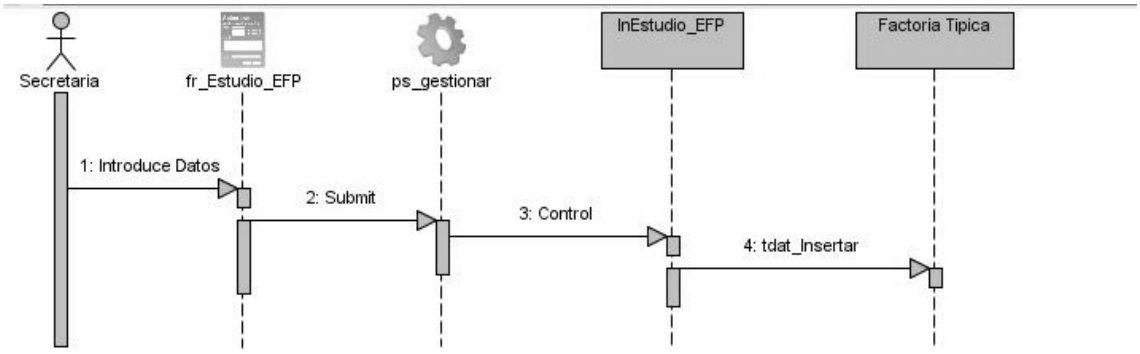


Figura 3.7. Vista de Diagrama de Secuencia con los objetos de Diseño.

Este Diagrama muestra los objetos del diseño, donde se hace una descripción del flujo de sucesos-diseño a la hora de insertar información para el CU Estudio del EFP.

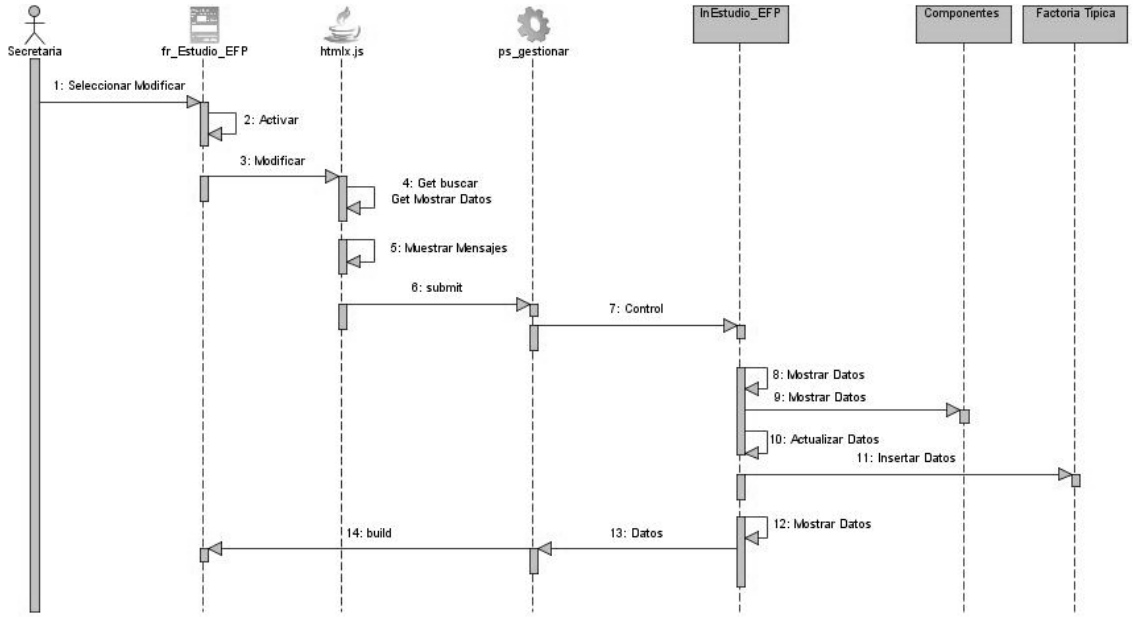


Figura 3.8. Vista de Diagrama de Secuencia con los objetos de Diseño.

Este Diagrama muestra los objetos del diseño, donde se hace una descripción del flujo de sucesos-diseño a la hora de mostrar alguna información para el CU Estudio del EFP.

3.4. Vista de Despliegue.

En esta sección se describe la configuración física de la red, o sea el hardware que necesita para que el Sistema sea instalado y corra. A continuación se muestra el diagrama de despliegue:

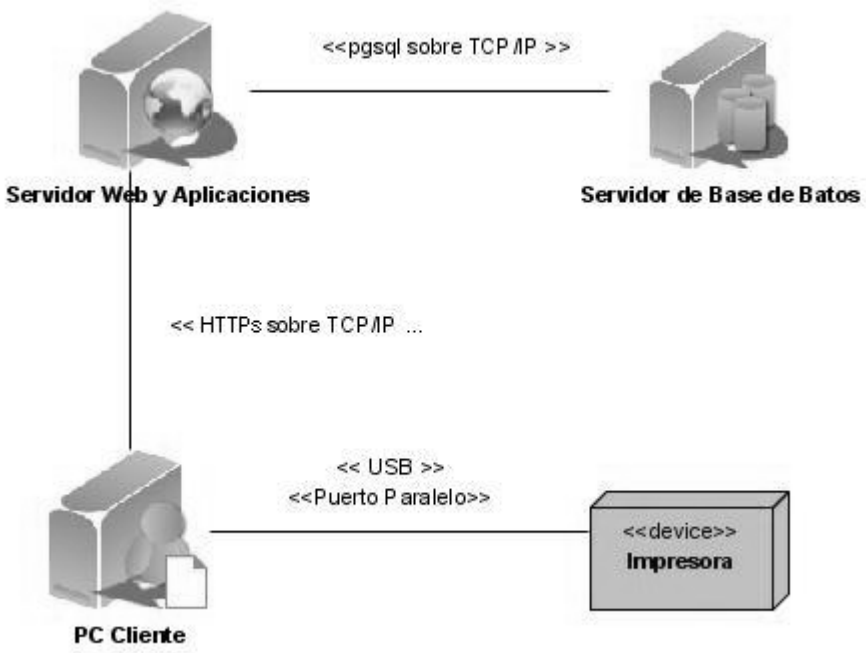


Figura 3.9. Vista de Despliegue.

Descripción de la configuración física de la red.

- PC Cliente:

Navegador (Internet Explorer *versión* 4.0 o superior, Mozilla Firefox *versión* 1.5 o superior.

- Servidor de Aplicación:

Tecnología Apache versión 2.0 o superior para el servidor Web.

Sistema operativo Windows 2000 o superior, o con sistema operativo UNIX (Linux).

- Servidor de Bases de Datos:

Gestor de Base de Datos PostgreSQL *versión* 8.0.

- Conexiones:

Protocolo *https* entre la PC cliente y el servidor de aplicaciones.

Protocolo puerto paralelo o USB entre la PC cliente y la impresora.

Protocolo TCP/IP entre el servidor de aplicaciones y el servidor de bases de datos.

3.5. Vista de Implementación.

En esta sección describe la estructura general del modelo de implementación, en correspondencia con la vista lógica, se debe representar la descomposición del software en capas y paquetes importante para la arquitectura.

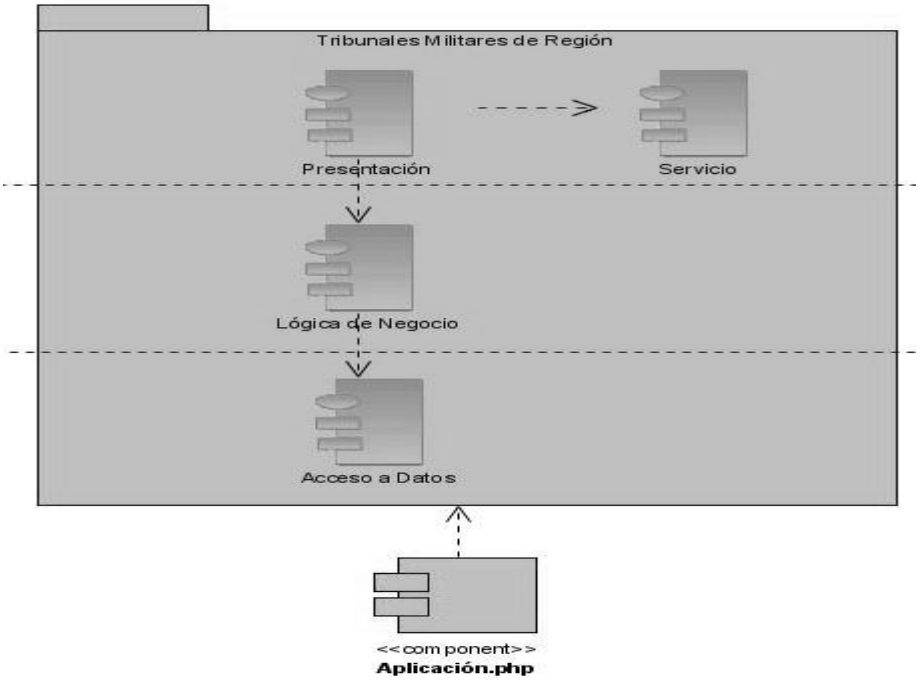


Figura 3.10. Vista General del Modelo de Implementación.

Cada uno, de dichos paquetes encapsulan uno o más componentes que se relacionan entre ellos para darle solución a la aplicación y todo depende de que esté instalada la Aplicación.php en los puestos de trabajo del cliente.

3.5.1. Descripción Global.

En esta subsección se dedica a definir cual es el contenido de cada capa, las reglas que definen si un componente debe ser parte de esa capa, y las interfaces entre capas, puede incluirse un diagrama de componentes que muestren la relación entre las capas.

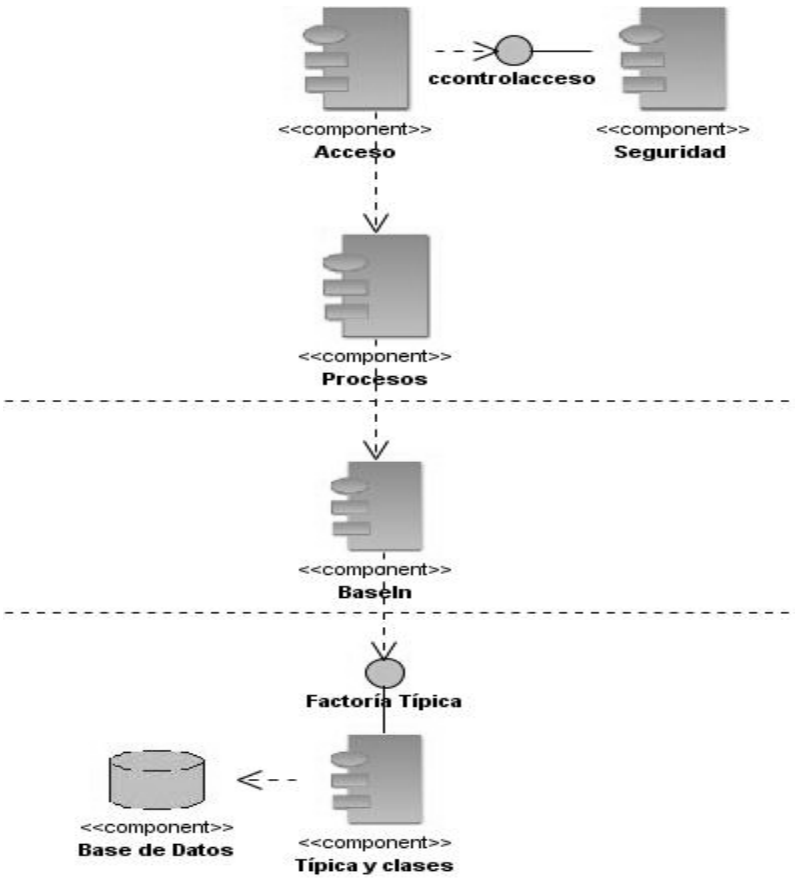


Figura 3.11. Vista del Diagrama de paquetes de componentes por capas.

Los componentes están distribuidos en las capas y paquetes de la Aplicación de la siguiente forma:

- **Capa de Presentación.**

- ✓ Paquete de componente de Acceso:

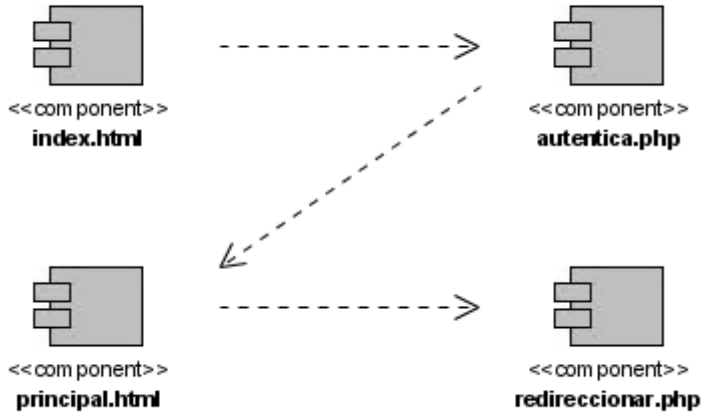


Figura 3.12. Vista del Diagrama de componentes de Acceso.

- ✓ Paquete de componente de Procesos :

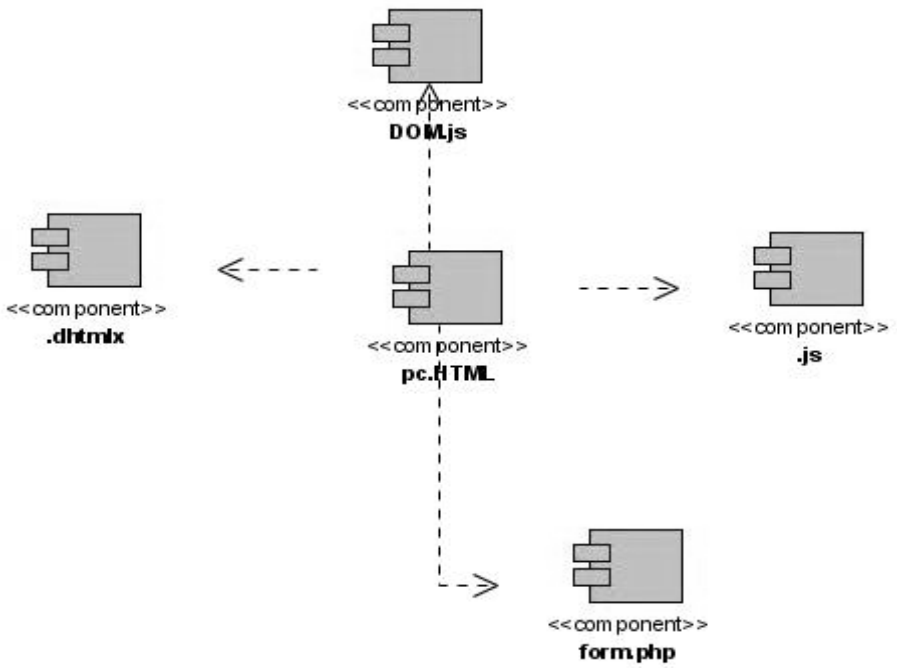


Figura 3.13. Vista del diagrama de Componentes de Procesos.

- **Capa de Lógica de Negocio.**

✓ Paquete de Baseln:

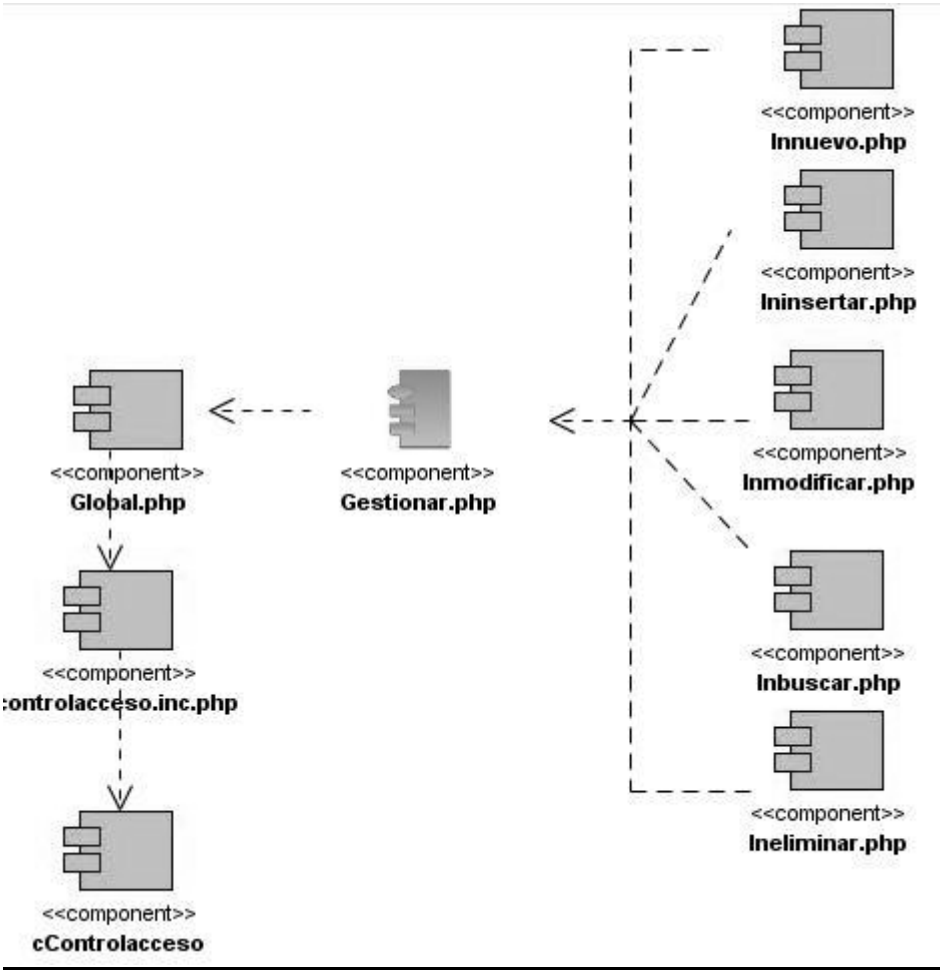
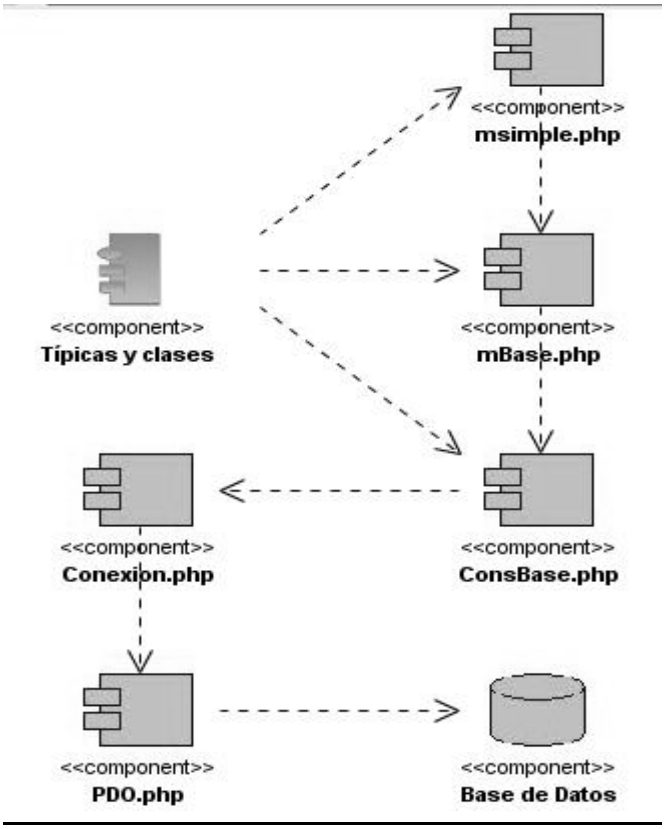


Figura 3.14. Vista del diagrama de Componentes de Baseln.

- **Acceso a Datos:**

✓ Paquete de Típicas y clases:



✓

Figura 3.15. Vista del diagrama de Componentes de Típicas y clases.

3.6. Calidad.

En esta sección se va realizar una descripción de cómo la Arquitectura del Software va a contribuir al desarrollo de todas las capacidades de la Aplicación en construcción. Antes de cumplir con el objetivo de esta sección se darán a conocer algunos términos para poder realizar esa evaluación.

3.6.1. Calidad Arquitectónica.

Barbacci et al. (1995) establecen que el desarrollo de formas sistemáticas para relacionar atributos de calidad de un sistema a su arquitectura provee una base para la toma de decisiones objetivas sobre acuerdos de diseño y permite a los ingenieros realizar predicciones razonablemente exactas sobre los atributos del sistema que son libres de prejuicios y asunciones no triviales. El objetivo de fondo es lograr la habilidad de evaluar cuantitativamente y llegar a acuerdos entre múltiples atributos de calidad para alcanzar un mejor sistema de forma global.

3.6.2. Atributos de Calidad.

Según Barbacci et al. (1995) la *calidad de software* se define como el grado en el cual el software posee una combinación deseada de atributos. Tales atributos son requerimientos adicionales del sistema (Kazman et al., 2001), que hacen referencia a características que éste debe satisfacer, diferentes a los requerimientos funcionales. Estas características o atributos se conocen con el nombre de atributos *de calidad*, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios (Barbacci et al. 1995).

3.6.3. Cualidades por la que puede ser evaluada una Arquitectura.

No es del todo cierto que se pueda decir que el sistema alcanzará todas sus metas de calidad con solo mirar la arquitectura. Pero muchos atributos de calidad yacen directamente en el centro de la arquitectura. Una arquitectura puede ser evaluada en base a los siguientes atributos de calidad:

- Seguridad: Es la medida de la habilidad del sistema de resistirse al uso no autorizado y negar los servicios, mientras los provee a usuarios legítimos.
- Modificabilidad: Es la habilidad de realizar cambios al sistema en forma rápida y a bajo costo.
- Portabilidad: Es la habilidad del sistema de correr sobre diferentes ambientes. Estos ambientes pueden ser de hardware, de software o una combinación de ambos. Portabilidad es un caso particular de modificabilidad.
- Funcionalidad: Es la habilidad del sistema de hacer el trabajo para el cual fue construido.

- La fiabilidad: Está relacionada con la capacidad para reaccionar ante fallos internos o externos que puedan afectarlo, en este sentido existen dos elementos fundamentales la madurez y la tolerancia a fallos. La madurez es la capacidad del producto software para evitar fallar como resultado de fallos en el software, o sea es la capacidad de respuesta del software ante anomalías del producto
- Variabilidad: Es la capacidad de la arquitectura de ser expandida o modificada para producir nuevas arquitecturas. Variabilidad es importante cuando la arquitectura se va a utilizar como piedra fundamental de toda una familia de productos relacionados, como ser una línea de producto.
- Susceptibilidad: Es la habilidad de soportar la producción de un subconjunto del sistema. susceptibilidad permite el desarrollo incremental. Es un tipo especial de Variabilidad.
- Integridad conceptual: Es la visión que unifica el diseño del sistema en todos los niveles. La arquitectura debe hacer cosas similares en forma similar. Debe mostrar consistencia en los mecanismos, decisiones y patrones aplicados.

3.6.4. Métodos de evaluación de arquitecturas de software.

Los Métodos de utilidad general para evaluar arquitecturas de software no existían hasta hace poco. En virtud de esto se han Propuesto múltiples métodos de evaluación tales como: Software Architecture Analysis Method (SAAM), Architecture Trade-off Analysis Method (ATAM), Active Reviews for Intermediate Designs (ARID), Cost-Benefit Analysis Method (CBAM), entre otros. La arquitectura de software posee un gran impacto sobre la calidad de un sistema, de ahí que es de vital importancia la evaluación de la arquitectura.

De acuerdo con Kazman et al. (2001) el método ARID es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura. Según los autores, ARID es un híbrido entre Active Design Review (ADR) y Architecture Trade-Off Method (ATAM), descrito anteriormente.

Según el planteamiento anterior el método ATAM (Método de Análisis de Acuerdos de Arquitectura) surge, del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros; esto es, los tipos de acuerdos que se establecen entre ellos.

El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. De cualquier forma, estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas, entre otros (Kazman et al., 2001).
Para evaluar la arquitectura mediante el método de ARID se deben tener en cuenta los atributos de calidad contemplados en la convivencia del diseño evaluado, ya que la etapa del proyecto en la que se aplica es a lo largo del diseño de la arquitectura.

3.6.5. Evaluación de la Arquitectura del Sistema.

La arquitectura de software propuesta para el Sistema de Tribunales Militares es adecuada ya que cumple con los atributos de calidad escogidos para la evaluación del sistema. El proceso que se lleva a cabo es iterativo e incremental y la evaluación de la AS se puede efectuar en cualquier etapa de la vida del desarrollo de la arquitectura, garantizando un profundo efecto en el sistema y en el proyecto. En principio el sistema está diseñado para soportar cambios de forma que no afecte a ninguna de las capas en que se dividida la arquitectura, dando la posibilidad de ser modificable y crecer en cualquier momento. La arquitectura debe ser funcional y cumplir con los requerimientos del usuario, aquí está presente la adecuada combinación de los estilos y patrones seleccionados que permitirán alcanzar esos requerimientos de calidad; otro atributo analizado es la fiabilidad, siendo imprescindible para el tratamiento de errores en el sistemas y la capacidad de respuesta ante fallos, estando presente dos características fundamentales como es la madurez y la tolerancia. También la arquitectura tiene definidos componentes en la capa de presentación que no permiten la entrada de cadenas no válidas, y en caso de que el usuario se valga de otros recursos para lograrlo, se le enseña a través de recursos visuales donde está el error y no procesará la información hasta que ésta sea entrada correctamente. La seguridad de acceso se tiene como un atributo muy importante a tener en cuenta en todo el desarrollo del sistema que no es más que la capacidad del producto software para proteger información y datos de manera que las personas o sistemas no autorizados no puedan leerlos o modificarlos, al tiempo que no se deniega el acceso a las personas o sistemas autorizados, este servicio lo ofrece un sistema de seguridad de los proyectos de UCI_MINFAR acoplado a la arquitectura diseñada, brindando seguridad a la base de datos contra usuarios no autorizados, puesto que se maneja información que hay que proteger en este sentido.
La portabilidad es un aspecto que siempre se tuvo en cuenta a la hora de desarrollar el sistema y las decisiones que se tomaron giraron entorno a la arquitectura. El sistema puede ser ejecutado en plataforma

Linux, teniendo como ventaja que una arquitectura en capas posibilita que se pueda hacer con mayor facilidad una migración a Linux, la migración puede que traiga algunos problemas, afectando algunas de las capas, pero esto no afectaría a las restantes capas. Otra de las decisiones tomadas fue la selección de la tecnología de programación, la metodología a utilizar para la descripción de la arquitectura, los servidores Web y la herramienta para confeccionar la *base de datos*; que harían que el sistema fuera un producto portable.

A grandes rasgos, estos atributos explicados y otros como la variabilidad, la susceptibilidad y la integridad conceptual que también están presentes, son los que posibilitaron la evaluación de la arquitectura con la calidad requerida. Teniendo en cuenta que la evaluación de la arquitectura no produce resultados cuantitativos, sino que nos dice si una arquitectura es adecuada a un conjunto de metas trazadas, se puede concluir que toda esta evaluación a ayudado a encontrar debilidades y ha mostrar un camino de riegos para ser solucionados antes de terminar el producto, siempre teniendo en cuenta que la condición principal es cumplir con los requerimientos del usuario.

3.7. Conclusión.

En este Capitulo luego de realizar una representación de las diferentes Vistas Arquitectónicas, permitiendo describir la arquitectura, se puede concluir, que las mismas son un subconjunto de modelo de diseño de software, incluyendo los elementos significativos de la arquitectura y excluyendo el diseño de los *componentes* básicos, resuelve a la hora de tomar decisiones sobre qué desarrollar y qué reutilizar, facilitándole un mejor desempeño y entendimiento para el resto de los desarrolladores que tienen como tarea de darle terminación al producto. Además, después de realizarle una evaluación según los atributos de calidad, haciendo necesarios que permita el funcionamiento de los requisitos del cliente durante el desarrollo del Sistema, se puede obtener una aplicación robusta y permisible a los cambios futuros.

CONCLUSIONES.

Siendo la Arquitectura uno de los elementos fundamentales para la construcción del software que se utilizará para la realización de todos los procesos judiciales de los Tribunales Militares de Región, por tanto, se ha diseñado la arquitectura basada en una tecnología Web, pero para poder cumplir con los objetivos trazados, primeramente se ha investigado sobre el sistema existente en cuanto a sus funcionalidades, verificando como se gestiona las información a la hora de informatizarlas, junto con los analistas del sistema, además se han realizado otras tareas como:

- Se ha hecho una investigación de los diferentes tipos de arquitecturas y patrones de diseño más factibles para mejorar el diseño del Sistema, en el caso de los patrones se tendrá que revisar en determinado momento durante el desarrollo del sistema, por los posibles cambios que pudieran ocurrir.
- Se definió la herramienta que permitió describir la Arquitectura.
- Se creó una documentación para la Arquitectura del Sistema, donde se describió y se evaluó la misma.
- Se definió la estrategia de trabajo mediante la línea base de la arquitectura y el marco de trabajo.

Además de cumplir con todo lo antes mencionado la arquitectura propuesta, y después de un análisis en correspondencia con el problema a resolver, se tendrán nuevas oportunidades para análisis profundos, incluyendo verificaciones de consistencia del sistema, conformidad con las restricciones impuestas por un estilo, conformidad con atributos de calidad, análisis de dependencias, entre otros. También se considera que la arquitectura propuesta se puede construir por parte del equipo de desarrollo sin ningún problema. Un punto muy importante que no debe dejar de mencionarse, es la reutilización de los diferentes componentes del sistema, escogidos todos de sistemas ya existentes de los proyectos de UCI_MINFAR, que facilitaron una buena estructuración del sistema que aquí se propone; tener en cuenta también como parte del ciclo de desarrollo, el constante refinamiento de la arquitectura de software.

RECOMENDACIONES.

- Ir incorporando los diferentes patrones que se vayan utilizando a medida que se desarrolle la construcción del sistema, en el documento de la arquitectura.

- Reutilizar la arquitectura de la primera versión del Sistema de los Tribunales Militares de Región a los sistemas de los niveles superiores.

REFERENCIAS BIBLIOGRÁFICAS.

Ivar Jacobson, Grady Booch, James Rumbaugh. 2004. *Proceso Unificado de Desarrollo de Software Volumen I.* 2004.

Pressman, Roges S. 2005. *Ingenieria de Software un Enfoque Práctico.* 2005.

Quesada, Ricardo Alarcon de. 2002. *Ley No 97 del Tribunal Militar.* Articulo 1, Ciudad de la Habana,Cuba : s.n., 2002.

—. 2002. Ley No 97 del Tribunal Militar. articulo 4.1, Ciudad de la Habana,Cuba : s.n., 2002.

—. 2002. Ley No 97 del Tribunal Militar. articulo 3 Inc. a) y b),, Ciudad de la Habana,Cuba : s.n., 2002.

Reynoso, Carlos Billy. 2004. Architect Academy: Seminario de Arquitectura de Software. [En línea] 2004. http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp.

2007. Software Engineering Institute . [En línea] Carnegie Mellon University, 2007. http://www.sei.cmu.edu/architecture/.

Escribano, Gerardo Fernández. 2002. Introducción a Extreme Programming. 2002.

Hernández, Pedro V. 2004-2005. *Unidad Docente de Ingeniería del Software.* [Online] 2004-2005. resumen de doctorado. http://is.ls.fi.upm.es/doctorado/Trabajos20042005/Hernandez.pdf.

Rodriguez, Angelo Roberto Rodriguez. 2006. Consultoria en Seguridad. *Creangel.com.* [Online] 2006. http://www.creangel.com/uml/intro.php.

BIBLIOGRAFÍA.

Reynoso, Carlos Billy. 2004. Architect Academy: Seminario de Arquitectura de Software. [En línea] 2004. http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp

Manual del profesor arquitectura v1.1 Conferencia 6 de la Universidad de las Ciencias Informáticas. Tema No.1

Escribano, Gerardo Fernández. 2002. Introducción a Extreme Programming. [Online] 2002. http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf.

RODRIGUEZ, ANGELO ROBERTO RODRIGUEZ. 2006. consultoria en seguridad. Lenguaje Unificado de Modelamiento (UML). [Online] 2006. http://creangel.com/uml/intro.php.

Racional Unified Process Version 2003.06.00.65.Copyright 1987 – 2003 Rational Software Corporation.

Doc. Evaluación de Arquitecturas de Software .Universidad de la República – Facultad de Ingeniería – Instituto de Computación

ERIKA CAMACHO, FABIO CARDESO, GABRIEL NUÑEZ. 2004 . ARQUITECTURAS DE SOFTWARE. GUÍA DE ESTUDIO. [Online] 2004 . http://prof.usb.ve/lmendoza/Documentos/PS-116/Guia%20Arquitectura%20v.2.pdf.

Ivar Jacobson, Grady Booch, Jame Rumbaugh. Editorial Felix Varela. 2004. Proceso Unificado de Desarrollo Software.Volumen I *y I.* la Habana : s.n., 2004.

Varela, Craig Larma. Editorial Felix. 2004. UML y Patrones 1 y 2.Introducción al Análisis y Diseño Orientado a Objetos . La Habana : s.n., 2004.

ANEXOS.

Estándares de Codificación:

Apariencia de clases y objetos	Primera letra en mayúscula	Los nombres de las clases y las instancias de las mismas deben comenzar con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación PascalCasing*. Ejemplo: MiClase ().
Nombre de clases y objetos	Relacionados al propósito	El nombre empleado, debe permitir que con sólo leerlo se conozca el propósito de la clase o instancia de la misma. Para el caso de las instancias es recomendable que se denoten así: Para la clase: Nomcliente su instancia será $Ocliente, de forma tal que la primera letra indique que es un objeto y el resto, la clase a la que pertenece.
Apariencia de atributos	Primera letra en minúscula	El nombre que se le da a los atributos de las clases debe comenzar con la primera letra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing**.
Nombre de atributos	Nemotécnicos	El nombre empleado, debe permitir que con sólo leerlo se conozca el propósito del mismo dentro de la clase. Ejemplo: $nTabla, este atributo denota el nombre de una tabla.
Apariencia de las funciones	Primera letra en mayúscula	Los nombres de las funciones deben comenzar con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación PascalCasing*. Si son funciones que obtienen un dato se emplea el prefijo get y si fijan algún valor se emplea el prefijo set.
Nombre de las funciones	Nemotécnicos	El nombre empleado, debe permitir que con sólo leerlo se conozca el propósito de la misma dentro de la clase.
Declaración de parámetro en funciones	Agrupados por tipos primero los string, los numéricos y	Los parámetros que se le pasan a las funciones se recomienda sean declarados de forma tal que estén agrupados por el

	valores por defecto.	tipo de dato que contienen. Ejemplo: BuscaUnidad ($nTabla(string),$nCampos(string),$kIndice (entero)).
	Variables y constantes	
Apariencia de constantes	Todas sus letras en mayúscula	Se deben declarar las constantes con todas sus letras en mayúscula.
Nombres de las variables y constantes	Nemotécnicos	El nombre empleado, debe permitir que con sólo leerlo se conozca el propósito de la misma. Ejemplo: $nFields.

Declaración de constantes y asignación a variables	Una por cada línea	Se recomienda declarar una constante por cada línea y con las asignaciones a las variables sucede lo mismo. Ejemplo:
		define("CONSTANT1","value1");
		define("CONSTANT2","value2");
		$nTabla='nomproducto';
		$kIndice=0;

	Identación	
Objetivo: Lograr una estructura uniforme para los bloques de código así como para los diferentes niveles de anidamiento.		
0 espacios en blanco desde la izquierda en	Require Include Class	No se empleará ningún espacio en blanco desde la izquierda para las instrucciones antes mencionadas. Se tomará como inicio de la página el tag PHP <?
2 espacio en blanco desde la izquierda en	Function Define	Se dejarán dos espacios en blanco desde la izquierda en las instrucciones antes mencionadas.
2 espacio en blanco desde la referencia en	Inicio y fin de bloque	Se recomienda dejar dos espacios en blanco desde la instrucción anterior para el inicio y fin de bloque {}. Lo mismo sucede para el caso de las instrucciones If, else, For, While, Do While, Switch, Foreach.
Niveles de anidación	Hasta 5 niveles	Se recomienda emplear hasta 5 niveles de anidación en instrucciones If, For, While.

Ejemplo de identación

```
<?
require ('class/Interface.php');

class MiClase
{
  function BuscaUnidad($nTabla, $nFields, $kIndice)
  {
   if ($nTabla)
   {
        …
```

```
    }
    for (…)
    {
      …
    }
  }
}
?>
```

Comentarios, separadores, líneas y espacios en blanco		
Objetivo: Establecer un modo común para comentar el código de forma tal que sea comprensible con sólo leerlo una vez.		
Ubicación de comentarios	Al inicio de cada clase o función y al final de cada bloque de código.	Se recomienda comentar al inicio de la clase o función especificando el objetivo de la misma así como los parámetros que usa (especificar tipos de dato, y objetivo del parámetro) entre otras cosas. Y se comenta también cuando se cierran los ciclos, clases, instrucciones if y otras.
Separador de instrucciones	Se emplea el punto y coma.	Se recomienda usar el separador al final de cada instrucción y no en la línea de abajo. Ejemplo: define (“CONSTANT”,”value1”);
Líneas en blanco	Se emplean antes de cada función.	Se recomienda dejar una línea en blanco antes de la definición de cada función para dar claridad al código.
Espacios en blanco	Entre operadores lógicos y aritméticos.	Se recomienda usar espacios en blanco entre estos operadores para lograr una mayor legibilidad en el código. Ejemplo: $nTabla = ‘nomproducto’; if (($nTabla) && ($nFields))

Tabla.1. Estándares de Código.

Estándares para la Base de Datos.

Apariencia de la BD	Primera letra en mayúscula	Los nombres de las BDs deben comenzar con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación PascalCasing*.
Nombres de las BDs	Nemotécnicos y relacionados al propósito.	El nombre empleado, debe permitir que con sólo leerlo se conozca el propósito de la misma.
Apariencia de los esquemas	Todas las letras en minúscula.	El nombre a emplear para los esquemas debe escribirse con todas las letras en minúscula para evitar problemas con el

		Case Sensitive del gestor. Ejemplo: create schema ‘finanzas’;
Nombres de los esquemas	Nemotécnicos y relacionados al propósito.	El nombre empleado, debe permitir que con sólo leerlo se conozca el propósito del mismo.
Apariencia de las tablas	Todas las letras en minúscula.	El nombre a emplear para las tablas debe escribirse con todas las letras en minúscula para evitar problemas con el Case Sensitive del gestor. Ejemplo: create table ‘nom_producto’;
Nombres de las tablas	Nemotécnicos y relacionados al propósito. Además clasificando las tablas por su tipo.	El nombre empleado, debe permitir que con sólo leerlo se conozca el propósito del mismo. Se deben clasificar las tablas por su tipo, es decir por los datos que contienen se le coloca un prefijo, que se puede clasificar en: Ejemplo: Nomencladores nom_… Auxiliares aux_... Datos dat_... Históricas his_... Seguridad seg_... Temporales tmp_... Configuración cfg_...
Apariencia de los campos	Todas las letras en minúscula.	El nombre a emplear para los campos debe escribirse con todas las letras en minúscula para evitar problemas con el Case Sensitive del gestor. Ejemplo: add field ‘idproducto’;
Nombre de los campos	Nemotécnicos En caso de identificadores, emplear id, este seria igual en la tabla de datos que lo emplea.	El nombre empleado, debe permitir que con sólo leerlo se conozca el propósito del mismo. Además se debe incluir un comentario en la descripción del mismo.
Nombre de las llaves primarias	Nemotécnicos empleando prefijos.	Se nombrarán las llaves primarias de forma que se vea de qué tabla es y que es primaria. Ejemplo:

		pk_cuenta. (Llave primaria de la tabla cuenta). Si es una llave compuesta se coloca el prefijo y en nemotécnico los campos que la forman.
Nombre de las llaves foráneas.	Nemotécnicos empleando prefijos.	Se nombrarán las llaves foráneas de forma que se vea de qué tabla es y que es foránea. Ejemplo: fk_cuenta. (Llave foránea de la tabla cuenta). Si es una llave compuesta se coloca el prefijo y en nemotécnico los campos que la forman.
Nombre de las secuencias	Nemotécnicos empleando prefijos.	Se nombrarán las secuencias de forma que se vea de qué campo es y que es una secuencia. Ejemplo: seq_idcuenta. (Secuencia del campo idcuenta).
Restricciones Únicas y de Chequeo	Nemotécnicos empleando prefijos.	Ejemplo: (u_ o c_) + nombre del campo que la emplea.
Nombres de las funciones, triggers, y vistas	Prefijos + Nemotécnicos	El nombre empleado, debe permitir que con sólo leerlo se conozca el propósito del mismo. Ejemplo: ft_ Funciones de triggers.

Tabla.2. Estándares de BD.

- *Notación PascalCasing: Los identificadores y nombres de variables, métodos y funciones están compuestos por múltiples palabras juntas iniciando cada palabra con letra mayúscula. Ejemplo: NotacionPascalCasing.
- **Notación CamelCasing: Los identificadores y nombres de variables, métodos y funciones están compuestos por múltiples palabras juntas iniciando cada palabra con letra mayúscula excepto la primera palabra que debe iniciar con minúscula. Ejemplo: notacionCamelCasing.

GLOSARIO.

*API***:** del inglés Application Programming Interface - Interfaz de Programación de Aplicaciones, es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.

Arquitecto de software: es el responsable de la arquitectura del software, que incluye las decisiones técnicas más importante en cuanto a las restricciones del diseño global e implementación del proyecto.

Base de datos: conjunto no redundante de información almacenada en memoria organizada independientemente de su utilización y su implementación en máquinas accesibles en tiempo real y compatible con usuarios concurrentes con necesidad de información diferente y no predicable en tiempo.

Browser o Navegador: aplicación para visualizar documentos WWW y navegar por Internet. En su forma más básica son aplicaciones hipertexto que facilitan la navegación por los servidores de navegación de Internet. Los más avanzados, cuentan con funcionalidades plenamente multimedia y permiten indistintamente la navegación por servidores WWW, FTP, Gopher, acceso a grupos de noticias, la gestión del correo electrónico, etc.

Baseline: es una instantánea del estado de todos los artefactos del proyecto, registrada para efectos de gestión de configuración y control de cambios.

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

Componentes: más conocido como, los componente de Software**,** que son todo aquel recurso desarrollado para un fin concreto y que puede formar solo o junto con otro/s, un entorno funcional requerido por cualquier proceso predefinido. Son independientes entre ellos, y tienen su propia estructura e

implementación. Si fueran propensos a la degradación debieran diseñarse con métodos internos propios de actualización.

Drivers: un controlador de dispositivo (llamado normalmente controlador, o, en inglés, *driver*) es un programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del hardware y proporcionando una interfaz -posiblemente estandarizada- para usarlo.

Encapsulamiento: es una característica de la programación orientada a objetos.El encapsulamiento consiste en ocultar los detalles de la implementación de un objeto, a la vez que se provee una interfaz pública por medio de sus métodos permitidos. También se define como la propiedad de los objetos de permitir acceso a su estado solamente a través de su interfaz o de relaciones preestablecidas con otros objetos.

eXtreme Programming: programación extrema**,** es un enfoque de la ingeniería de software formulado por Kent Beck, autor del primer libro sobre la materia, es la más destacada de los procesos ágiles de desarrollo de software

FTP: (File Transfer Protocol) es un protocolo de transferencia de ficheros entre sistemas conectados a una red TCP basado en la arquitectura cliente-servidor, de manera que desde un equipo cliente se puede conectar a un servidor para descargar ficheros desde él o para enviarle los archivos independientemente del sistema operativo utilizado en cada equipo.

Herencia: es uno de los mecanismos de la programación orientada a objetos, por medio del cual una clase se deriva de otra de manera que extiende su funcionalidad. Una de sus funciones más importantes es la de proveer Polimorfismo y late binding.

Heterodoxos: es un método utilizado en la arquitectura a la hora de desarrollar un Software, es decir un método ágil.

HTTP: protocolo de transferencia de hipertexto (HTTP*, HyperText Transfer Protocol*) es el protocolo usado en cada transacción de la Web (WWW). El hipertexto es el contenido de las páginas web, y el protocolo de

transferencia es el sistema mediante el cual se envían las peticiones de acceso a una página y la respuesta con el contenido.

IEEE: corresponde a las siglas de *The Institute of Electrical and Electronics Engineers*, el Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización.

Indirección: es una técnica de programación. El concepto se basa en hacer referencia indirecta a los datos usando las direcciones de memoria que los contienen o mediante punteros que señalan hacia esos datos o a las direcciones que los contienen.

Ingeniería del Software: es el estudio de los principios y metodología para desarrollo mantenimiento de de sistema de software.
ISO: la Organización Internacional para la Estandarización o International Organization for Standarization (ISO), es una organización internacional no gubernamental, compuesta por representantes de los organismos de normalización (ONs) nacionales, que produce normas internacionales industriales y comerciales. Dichas normas se conocen como normas ISO

JavaScript: es un lenguaje interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C.

JavaBeans: son un modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones en Java.

Microsoft: (acrónimo de Microcomputer Software), es una empresa de Estados Unidos, fundada por Bill Gates y Paul Allen, los cuales siguen siendo sus principales accionistas. Dueña y productora de los sistemas operativos: Microsoft DOS y Microsoft Windows, que se utilizan en la mayoría de las computadoras del planeta.

Mapeo objeto-relacional: (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R 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.

Ontología: En filosofía ciencia, estudio, teoría) o Metafísica general es el estudio de lo que es en tanto que es y existe. Por ello es llamada la teoría del ser, es decir, el estudio de todo lo que es: qué es, cómo es y cómo es posible. La Ontología se ocupa de la definición del ser y de establecer las categorías fundamentales o modos generales de ser de las cosas a partir del estudio de sus propiedades.

OMT: Organización Mundial del Turismo que es creada en 1925 con el propósito de promover el turismo. Vincula formalmente a las Naciones Unidas desde 1976 al transformarse en una agencia ejecutiva del PNUD.

OMG: el Object Management Group (de sus siglas en inglés *Grupo de Gestión de Objetos*) es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA.

Paradigmas: representa un enfoque particular o filosofía para la construcción del software.

PDAs: del inglés *Personal Digital Assistant*, (Ayudante personal digital) es un computador de mano originalmente diseñado como agenda electrónica (calendario, lista de contactos, bloc de notas y memos) con un sistema de reconocimiento de escritura.

Polimorfismo: se denomina a la capacidad que tienen objetos de diferentes clases de responder al mismo mensaje en programación orientada a objetos.

RPC: (del inglés Remote Procedure Call, Llamada a Procedimiento Remoto) es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos.

RMI:(Java Remote Method Invocation, Java la Invocación del Método Remota) es un mecanismo ofrecido en Java para invocar un método remotamente.

Scrum: es un modelo para el desarrollo de productos tecnológicos, también se emplea en entornos que trabajan con requisitos inestables y que requieren rapidez y flexibilidad; situaciones frecuentes en el desarrollo de determinados sistemas de software.

Servicios monolíticos: es un conjunto de actividades consistentes que buscan responder a una o más necesidades de un cliente.

Smalltalk: es un sistema informático que permite realizar tareas de computación mediante la interacción con un entorno de objetos virtuales. Metafóricamente, se puede considerar que un Smalltalk es un mundo virtual donde viven objetos que se comunican mediante el envío de mensajes.

SMTP: Simple Mail Transfer Protocol (SMTP), o protocolo simple de transferencia de correo electrónico. Protocolo de red basado en texto utilizado para el intercambio de mensajes de correo electrónico entre computadoras o distintos dispositivos (PDA's, teléfonos móviles, etc.). Está definido en el RFC 2821 y es un estándar oficial de Internet.

SOAP: (siglas de *Simple Object Access Protocol*) es un protocolo estándar creado por Microsoft, IBM y otros, está actualmente bajo el auspicio de la W3C que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. SOAP es uno de los protocolos utilizados en los servicios Web.

UDDI: son las siglas del catálogo de negocios de Internet denominado *Universal Description, Discovery and Integration*. El registro en el catálogo se hace en XML. UDDI es una iniciativa industrial abierta (sufragada por la OASIS) entroncada en el contexto de los servicios Web.

Versión: en software, es un número que indica el nivel de desarrollo de un programa. Es habitual que una aplicación sufra modificaciones, mejoras o correcciones. El número de versión suele indicar el avance de los cambios.

WSDL: son las siglas de *Web* Services Description Language, un formato XML que se utiliza para describir servicios Web.

XML: sigla en inglés de eXtensible Markup Language (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos.