Universidad de las Ciencias Informáticas

Facultad 6

Título:

"Definición de un Entorno Tecnológico de Desarrollo en la Línea de Integración de Soluciones".

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autora: Naridian Nuñez Borges

Tutor(es): Ing.Dioleysis Fontela González

Ing. Oscar Reimar Cedeño Delgado

La Habana. Junio 2011

"No se vive celebrando victorias, sino superando derrotas."

Ernesto Ché Guevara

DECLARACIÓN DE AUTORÍA

Declararo que soy autora de la presente investigación**:** **Definición de un Entorno Tecnológico de Desarrollo en la Línea de Integración de Soluciones** y reconozco, a la Universidad de las Ciencias Informáticas, sus derechos patrimoniales sobre la misma con carácter exclusivo.
Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Naridian Nuñez Borges.

_____

Firma de la Autora

Ing.Dioleysis Fontela González

_____

Firma del Tutor

. Ing.Oscar Reimar Cedeño.

_____

Firma del Tutor

DATOS DE CONTACTO

Tutor: Ing. Dioleisys Fontela González.

e-mail:dfontela@uci.cu

Tutor: Ing. Oscar Reimar Cedeño Delgado

e-mail:orcedeno@uci.cu

AGRADECIMIENTOS

Mamá eres la única persona del mundo que siempre está, de forma incondicional. Si te rechazo, me perdonas. Si me equivoco, me acoges. Si los demás no pueden conmigo, me abres una puerta. Si estoy feliz, celebras conmigo. Si estoy triste, no sonríes hasta que me hagas reír. Gracias Tita, por amarme tanto, por llenar mi vida con tanta felicidad y ser parte de este sueño hecho realidad.

Gracias papá por todos los regaños que me hicieron la persona que soy. Gracias porque después de mis errores y tristezas siempre estuviste con los brazos abiertos para consolarme. Gracias porque en mis grandes triunfos también estuviste para disfrutar conmigo las grandes felicidades.

Los Quiero Mucho

A mi hermano Angel Luis , gracias por darme tu cariño, apoyo, por estar siempre a mi lado y ayudarme tanto.

A mi hermanita del alma Ceci, mil gracias mi amiga por existir, por ser cómplice de cada momento de mi vida. Tú amistad es como ese travieso duende que anda de bosque en bosque en busca de alegrías, tú confianza hacia mí siempre ha sido tan grande como el inmenso mar. Tú cariño, tu apoyo y dedicación se convirtieron en el paso de los años en esa pequeña lucecita que ilumina los senderos más oscuros de mi

vida. Recuerda mi amiga nuestra amistad no se mide en minutos ni en años. Se mide en lealtad, comprensión y muchos momentos de felicidad.

Titi Nada se compara Contigo

A mi amigo Alain por su, amistad, confianza y todas esas palabras de aliento que supo brindarme cuando las necesitaba.

A mis tutores Oscar y Dioleysis por todo el apoyo que supieron brindarme durante el desarrollo de esta investigación.

Gracias a todas aquellas personas que de una forma u otra me brindaron su cariño, su ternura, su amor, iluminando mis días con su alegría.

DEDICATORIA

A mis padres que han tenido el papel protagónico en la mujer que soy y seré toda la vida.

Mamá te agradezco por todo lo que has hecho en la vida por mí, tus brazos siempre se abrían cuando quería un abrazo, tu corazón comprendía cuando necesitaba una amiga. Tus ojos tiernos se endurecían cuando me hacía falta una lección. Tu fuerza y tu amor me guiaron, y me dieron alas para volar. Muchas gracias tita de mi vida por tu amor, confianza, dedicación, apoyo, por ser una madre ejemplar, excepcional y lo que más quiero de esta vida.

A mi padre por engendrarme todo el respeto, el amor y el cariño que necesitaba para vencer cualquier obstáculo durante estos cinco años.Mil gracias papá por llenarme de argumentos para enfrentar la vida, hoy quiero decirte que acabó la carrera de los 400 metros, ahora comienza una mucho más larga y compleja.

Son ustedes la razón de mi existir y la luz de mi vida, los Amo.

RESUMEN

En el presente Trabajo de Diploma se describe una Arquitectura de Software(AS) que podría ser utilizada como referencia en el desarrollo de Aplicaciones Web. La investigación surge producto de la necesidad de un Entorno Tecnológico para el desarrollo en la línea de Integración de Soluciones. La arquitectura definida provee los elementos necesarios para el desarrollo de software y la posterior incorporación de nuevas funcionalidades dentro de la línea. Su éxito viene dado por las decisiones arquitectónicas que fueron tomadas a lo largo del ciclo de vida del desarrollo del producto, por la definición de la plataforma tecnológica, de los lineamientos y mecanismos arquitectónicos. Para la selección de las herramientas informáticas y tecnologías para el desarrollo del sistema se tuvo en cuenta la situación del país en materia de acceso a las herramientas propietarias y se decidió seguir la línea de aquellas que estuvieran bajo licencia libre. Se realizó una evaluación de la arquitectura definida aplicando algunos de los métodos de evaluación existentes de la arquitectura obteniendo resultados satisfactorios. También se creó un Catálogo de productos a través de los activos existentes en la línea, garantizando mayores posibilidades de reutilización, así como soluciones robustas que carezcan de complejidad innecesaria y con mejoras en propiedades como la escalabilidad y la reusabilidad. El presente trabajo constituye una guía para los desarrolladores que deseen emplear la combinación del framework de desarrollo Symfony con el framework de presentación ExtJS.

Palabras Clave

Integración de Soluciones, Aplicación Web, Arquitectura de Software (AS), Entorno Tecnológico, Framework, Catálogo.

Tabla de Contenido

INTRODUCCIÓN

En el mundo actual el desarrollo acelerado de las tecnologías, las comunicaciones y la informática han conllevado a un gran avance del desarrollo del software, que junto con las necesidades de la humanidad han permitido que estas se encuentren cada vez más presente en el entorno de trabajo. Debido a este constante y profundo desarrollo que ha existido es fundamental mencionar que la Arquitectura de Software ofrece una serie de servicios a los arquitectos y desarrolladores encaminados a facilitar la implementación de aplicaciones empresariales, ya que esta se encuentra diseminada en el mundo con gran polarización en el consenso conceptual logrando de esta manera que el desarrollo de sistemas sea más productivo. La arquitectura de los sistemas, al ser la plataforma base de lo que se quiere construir, constituye el pilar fundamental de los sistemas informáticos, de las decisiones arquitectónicas correctas que se tomen y de las tecnologías más adecuadas que se decidan en el proceso de definición del producto depende en gran medida el desempeño posterior del mismo.

Otro aspecto que se considera fundamental dentro de la Arquitectura de Software son los activos que constituyen la base de una línea de arquitectura, siendo también el elemento fundamental de reutilización.

A pesar de la cantidad de definiciones que existen de Arquitectura de Software, la gran mayoría coincide en que esta se refiere a la estructura a grandes rasgos del sistema, estructura consistente en componentes y relaciones entre ellos. Por lo que se define Arquitectura de Software como la estructura y organización de los elementos de software de un Sistema Informático, cómo y bajo qué condiciones y configuraciones se van a comunicar dichos elementos, ya que la misma es una disciplina independiente de cualquier metodología de desarrollo, por lo que representa un alto nivel de abstracción del sistema y responde a los requerimientos no funcionales, pues estos se satisfacen mediante el modelado y diseño de la aplicación (1).

Cuba no está ajena a estos cambios existentes dentro de dicha disciplina ni queda exenta del desarrollo acelerado de la ciencia y las tecnologías de la información, nuestro país se ha hecho sentir en este proceso y para ello se han dedicado esfuerzos al desarrollo de la informática en numerosas instituciones. Ello implica que las empresas informáticas enfrenten cada día un reto para brindar una respuesta rápida, eficaz y con calidad a los clientes. Una muestra de estas entidades productoras de software la constituye la Universidad de las Ciencias Informáticas (UCI), que a pesar de ser un centro destinado a formar profesionales altamente calificados en esta rama, produce software y brinda

servicios informáticos. Si bien es conocido que la Arquitectura de Software no se estudia como disciplina en ninguno de los programas de la carrera, la formación en la misma es producto del auto aprendizaje de los especialistas, de aquí que su implementación práctica en la UCI no logre cubrir, normar y dirigir todos los aspectos integrados del desarrollo de la misma. En este caso, la poca experiencia de la disciplina en la UCI ha implicado que se incurra en costosos errores en el desarrollo y en muchos casos la afectación de los atributos de calidad de los productos o soluciones que en ella se desarrollan.

La UCI cuenta con una estructura de producción muy novedosa ayudando con ello al desarrollo de productos informáticos destinados a un gran número de esferas de la sociedad, tales como: salud, educación, deporte, turismo, biotecnología, telecomunicaciones, entre otras. La misma cuenta con varios centros de producción y desarrollo de software, los cuales se encuentran hoy en un gran proceso de avance e integración con distintos países hermanos. Una de estas entidades mencionadas es el Centro de Desarrollo de Tecnologías de Gestión de Datos (DATEC), dicho centro tiene como objetivo el desarrollo de herramientas y soluciones de gestión y análisis de datos. El mismo está dividido en diferentes líneas de trabajo entre las que se destaca Integración de Soluciones, la cual está destinada a desarrollar activos que tributan a un Paquete de Herramientas para la Ayuda a la Toma de Decisiones (PADTSI), donde la mayoría de estos activos presentan una arquitectura común. En la línea de Integración de Soluciones no se cuenta con una plataforma tecnológica, por lo que resulta necesario y es prioridad de la dirección del centro definir para la línea un entorno tecnológico donde se enmarquen las metodologías, herramientas y tecnologías a utilizar como ambiente de desarrollo y tributen a una conceptualización de un modelo de desarrollo de software. Por otra parte tampoco se cuenta con un catálogo de productos que describa las características de cada activo desde el punto de vista arquitectónico, ni con una propuesta de integración de los mismos en la construcción de nuevos productos y en la solución de situaciones que surgen durante el proceso de desarrollo de software, las cuales limitan y dificultan el proceso entre sí, tales como: la escasa reutilización, la mala selección de elementos de software, poco entendimiento por parte del equipo acerca de lo que se está desarrollando, mala asignación de funcionalidad a elementos del diseño, herramientas y framework, influyendo todo esto en cuestiones de escalabilidad, rendimiento y costo. Debido a la situación problemática anteriormente expuesta surge el siguiente:

Problema de la investigación: ¿Como contribuir a la mejora de la escalabilidad y rendimiento en el proceso de desarrollo de software en la línea de productos Integración de Soluciones?

La investigación tiene como **Objeto de estudio:** Arquitectura de Software, enmarcado en el **Campo de acción**: Arquitectura de Software para la línea de productos Integración de Soluciones.
Como **Objetivo general:** Definir un Entorno Tecnológico de Desarrollo que permita construir la línea base de las soluciones de software en la línea de productos Integración de Soluciones.

En correspondencia con ello, se plantean como **Objetivos específicos:**

1. Identificar elementos arquitectónicos de la línea de Integración de Soluciones.
2. Definir un entorno tecnológico de desarrollo para la línea de productos Integración de Soluciones.
3. Definir catálogo de productos de la línea de productos Integración de Soluciones.

Para dar cumplimiento a los objetivos se han trazado las siguientes **Tareas de investigación:**

- Realización de un estudio bibliográfico sobre Arquitectura de Software y Líneas de Productos de Software.
- Identificación de los elementos arquitectónicos básicos de los activos de la línea.
- Selección de las herramientas, metodología y tecnologías necesarias a utilizar en base a la solución del problema.
- Revisión de toda la documentación referente a las estrategias y restricciones que poseen los activos desarrollados en la línea de Integración de Soluciones para la definición del catálogo de productos.

Estructura de la Tesis

Capítulo 1 Fundamentación Teórica: En este capítulo se realiza la definición del modelo teórico de la investigación. También se analizará la Arquitectura de Software como disciplina dentro del proceso de desarrollo de software, sus tendencias actuales y evolución, así como una fundamentación de las

líneas de productos de software. También se analizarán aspectos sobre la Ingeniería de Software Basada en Componentes, y cómo la misma tiene una relación importante con la arquitectura de software.

El Capítulo 2 Tecnologías y Patrones: Éste capítulo es el resultado de la búsqueda y el análisis de la información correspondiente al proceso de diseño de la arquitectura del sistema. También se realizará un profundo análisis de los estilos arquitectónicos y patrones de arquitectura que se ajustan a la solución adecuada; además se definirá el marco de trabajo o framework que dará soporte a la aplicación, como las herramientas, metodología y tecnologías informáticas necesarias para su desarrollo y posterior explotación.

El Capítulo 3 Definición del catálogo de productos: En este capítulo se describe la arquitectura de referencia para la línea de productos de Software Integración de Soluciones, así como una definición conceptual de la línea misma y su estrategia de producción .Además se procede a la creación conceptual del modelo de referencia para el desarrollo de un Software y se describe cómo construir un producto a partir de los activos y de esta manera contribuir a la elaboración del catálogo de productos de la línea.

CAPITULO1: FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se realiza la definición del modelo teórico de la investigación. También se analizará la Arquitectura de Software como disciplina dentro del proceso de desarrollo de software, sus tendencias actuales y evolución, así como una fundamentación de las líneas de productos de software. También se analizarán aspectos sobre la Ingeniería de Software Basada en Componentes, y cómo la misma tiene una relación importante con la arquitectura de software.

1.1 Arquitectura de Software

La Arquitectura de Software es una práctica de apenas unos pocos años de trabajo constante. Tiene sus orígenes en la década del 60, pero no fue hasta los años 90 que Dewayne Perry y Alexander Wolf, la exponen en el sentido que hoy se conoce.
La Arquitectura de Software nace como disciplina del desarrollo de software específicamente en la década del 90. En 1992 en el trabajo de Dewayne Perry y Alexander Wolf “Foundations for the study of software Architecture“, donde por primera vez se habla del término “arquitectura”. En este trabajo ellos definen un modelo basado en la intuición que consta de 3 componentes: elementos, forma y razón. Los elementos pueden ser de procesamiento, de datos o de conexión. La forma define las propiedades de los elementos, sus relaciones y las restricciones que operan sobre estos. La razón proporciona los motivos subyacentes para una arquitectura determinada en términos de restricciones que derivan de los requisitos del sistema. (1).

A pesar de la cantidad de definiciones existentes sobre la disciplina Arquitectura de software en el año 2000 se publica la versión más reconocida por la recomendación IEEE 1471-2000 en la cual se expresa lo siguiente: *“La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución”.*

Es importante tener en cuenta que ésta es una de las definiciones más reconocidas sobre Arquitectura de Software además de ser la que se adopta en la línea , por lo que se tiene como objetivo

fundamental lograr homogenizar y ordenar la nomenclatura de descripción arquitectónica reconociendo a los estilos como el modelo fundamental de representación conceptual ya que la arquitectura en los proyectos de desarrollo de software constituye en la actualidad un tema extremadamente polémico; debido a que una incorrecta e inadecuada definición de la arquitectura de software puede llevar al caos a cualquier proyecto. En este sentido la línea de Integración de Soluciones trabaja por lograr que el desarrollo de los diferentes proyectos productivos sea satisfactorio y así de esta manera poder obtener productos con la calidad requerida y que cumplan con las expectativas de los clientes que solicitan el software. Para alcanzar las metas trazadas resulta necesario obtener en el grupo una estandarización e integrar cada uno de los equipos de trabajo existentes actualmente en la línea y de esta manera lograr que se pueda reutilizar la mayor cantidad de código posible, que se obtengan componentes altamente integrados y evitar que se realicen acciones innecesarias que provoquen retrasos en los proyectos, todo esto a través de la definición de una arquitectura sólida, robusta y bien concebida para el desarrollo de excelentes productos de software en la línea de Integración de Soluciones.

Ahora si bien se han analizado los elementos que dan lugar a la disciplina a nivel mundial y en la Línea de Productos de Software Integración de Soluciones, es necesario también tener en cuenta el factor humano interpretado en el rol del arquitecto, el cual se trata a continuación.

1.2 El Rol del Arquitecto de Software

El rendimiento del rol de arquitecto de sistema requiere conocimiento de las diversas disciplinas que contribuyen a la ingeniería de software y que posee potentes habilidades de análisis, razonamiento e inferencia, disciplina de autoestudio y superación, así como la habilidad de síntesis (2). Se ajusta a esta descripción el pensamiento que preside la definición del rol en la ayuda de Rational Unified Process (RUP):

"El arquitecto ideal debe ser una persona de letras, matemático, familiarizado con la historia, diligente estudioso de la filosofía, conocedor de la música, no ignorante de la medicina, entendido en las respuestas de las leyes, versado en la astronomía y en los cálculos astronómicos."

—Vitruvius, alrededor del25 a.C.

Se hace referencia aquí a un arquitecto de edificios pero sintetiza el hecho de que un arquitecto es en principio alguien con elevados niveles de preparación. El arquitecto de sistema no se ocupa simplemente de la tecnología de la solución, sino de muchas otras cuestiones como, por ejemplo, operación del sistema, rendimiento, viabilidad económica, capacidad de fabricación y soporte logístico, así como los factores políticos, técnicos, sociales, financieros y medioambientales que son clave para estas cuestiones. Se debe poseer la experiencia y madurez necesarios para permitir un análisis objetivo e intercambiar estudios que se van a realizar a fin de seleccionar la mejor solución, con la capacidad de discernir correctamente cuándo la información es incompleta o ambigua, y la capacidad de reconocer que lo mejor suele estar dictado por consideraciones políticas y económicas, además de consideraciones de ingeniería (2). Al trabajar con sistemas que están compuestos de personas, hardware y software, el arquitecto de sistema tiene que ser muy consciente de las limitaciones y restricciones físicas existentes en cualquier solución por los componentes humanos y de hardware. Además de la experiencia en la ingeniería de sistemas (y el conocimiento de disciplinas aliadas como, por ejemplo, la investigación de operaciones y la economía de ingeniería), el arquitecto de sistema debe tener unos sólidos conocimientos de ingeniería de software (a causa de su ubicuidad en los sistemas modernos), así como:

- **Experiencia**: en el dominio de problemas y un profundo conocimiento de los requisitos. Esta experiencia puede ampliarse mediante un equipo de arquitectura de sistemas.

- **Cualidades de liderazgo**: para dirigir el esfuerzo técnico entre los diferentes equipos, tomar decisiones críticas bajo presión y adherirse a estas decisiones. Para ser efectivos, el arquitecto de sistema y el gestor de proyectos deben trabajar conjuntamente, con el arquitecto de sistema dirigiendo las cuestiones técnicas y el gestor de proyectos dirigiendo las cuestiones administrativas. El arquitecto de sistema debe tener autoridad para tomar decisiones técnicas.

- **Habilidades**: de comunicación para ganar confianza, persuadir, motivar y orientar. El arquitecto de sistema no tiene autoridad en virtud de su cargo, sino a causa de su habilidad demostrada y sus resultados. Para ser efectivo, el arquitecto de sistema debe ganarse el respeto del equipo

de proyecto, el gestor de proyectos, el cliente y la comunidad de usuarios, así como del equipo de gestión.

El arquitecto de sistema es la fuerza técnica dirigente detrás del proyecto, no un consultor o un soñador. La carrera de un arquitecto de sistema que alcanza el éxito es una larga serie de decisiones sub-óptimas efectuadas en condiciones de incertidumbre y bajo presión. En la función de Arquitecto Principal debe tener conocimiento de áreas de la arquitectura como Seguridad, Acceso a Datos, Presentación, Negocio, Interacción con Periféricos, entre otras. El rol de arquitecto de sistema abarca las habilidades del diseñador de sistemas, aunque con un enfoque estratégico, no detallado.

A continuación se muestra una figura con las principales características del arquitecto de software dentro del sistema.

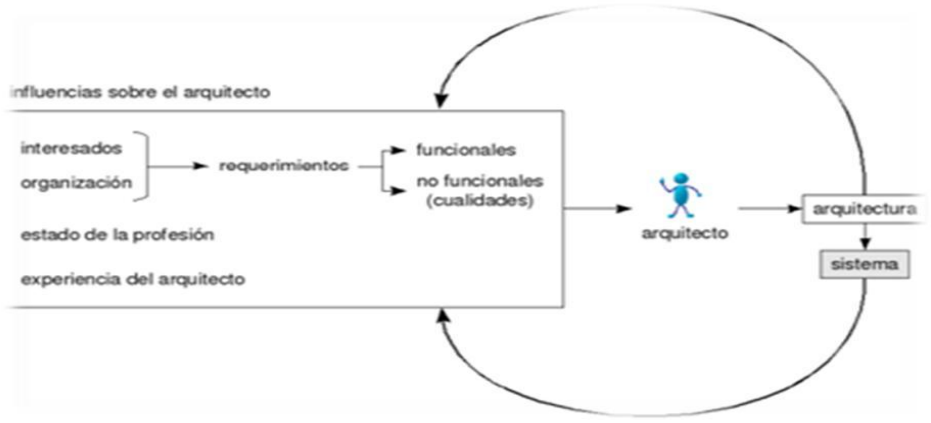


Figura # 1 El rol del Arquitecto de Software

1.3 La Calidad en la Arquitectura de Software

La arquitectura del software ha pasado a ser un artefacto esencial dentro del proceso de desarrollo de software moderno debido a sus múltiples usos, pero llegar a construir una arquitectura para el software que sea a la vez apropiada para dar cabida a las exigencias de las distintas partes interesadas y buena en términos absolutos es una tarea que dista muchísimo de ser sencilla.

La calidad arquitectónica es un elemento determinante en la calidad de software. Además es importante tener en cuenta 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 realizar predicciones razonablemente exactas sobre los atributos del sistema que son libres de prejuicios y asunciones no triviales.

A continuación se establece una clasificación de los atributos de calidad en dos categorías:

Observables vía ejecución (Externos): aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución.

No observables vía ejecución (Internos): aquellos atributos que se establecen durante el desarrollo del sistema.

- Dentro de los atributos observables se encuentran:

Disponibilidad (Availability): Es la medida de disponibilidad del sistema para el uso.

Funcionalidad (Functionality): Habilidad del sistema para realizar el trabajo para el cual fue concebido.

Desempeño (Performance): Grado en el cual un sistema o componente cumple con su funcionalidad, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria.

- Dentro de los atributos no observables se encuentran:

Integrabilidad (Integrability): Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.

Interoperabilidad (Interoperability): Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema.

Portabilidad (Portability): Es la habilidad del sistema para ser ejecutado en diferentes ambientes de cómputo.

Escalabilidad (Scalability): Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental.

Hacer un adecuado balance entre los distintos atributos de calidad es esencial para obtener un sistema que cumpla las expectativas de las distintas partes interesadas, pero el diseño de una arquitectura que tenga en cuenta los atributos deseados, sólo permitirá que el sistema resultante tenga estas características, pero no lo garantiza (3).

Si las decisiones arquitectónicas determinan los atributos de calidad del sistema, entonces es posible evaluar las decisiones arquitectónicas con respecto a su impacto sobre dichos atributos. A continuación se relacionan algunos métodos para la evaluación de la arquitectura.

1.4 Métodos de evaluación de arquitecturas de software.

Los métodos de evaluación sirven de guía a los involucrados en el desarrollo del sistema para la búsqueda de conflictos que pueden presentar una arquitectura y sus soluciones .Seguidamente se abordan algunos de los métodos:

El Método de Análisis de Arquitecturas de Software (SAAM)

El Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como: modificabilidad, portabilidad, escalabilidad e Integrabilidad. Se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro.

El Método de Análisis de Acuerdos de Arquitectura (ATAM)

El Método de Análisis de Acuerdos de Arquitectura (Architecture Trade-off Analysis Method, ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM. El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados, los cuales 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 (2).

Método de Evaluación para Arquitecturas Basadas en Componentes (MECABIC)

El objetivo principal de MECABIC es evaluar y analizar la calidad exigida por los usuarios sobre Arquitectura de Software Basadas en Componentes (ASBC). Además este permite centrarse en características que dependen exclusivamente de la arquitectura y al ser un estándar facilita la correspondencia con características de calidad (4).

La selección del método basado en arquitectura se realizó a través de una comparación entre los métodos estudiados: SAAM, ATAM y MECABIC analizando aspectos relevantes, donde se concluyó que el método SAAM se sugiere cuando el atributo de calidad modificabilidad es el de mayor interés. Mientras que el método ATAM es más profundo para evaluar aspectos relacionados con la arquitectura, como el desempeño o la confiabilidad. Por otro lado el MECABIC tiene como objetivo principal evaluar y analizar la calidad exigida por los usuarios sobre Arquitectura de Software Basadas en Componentes (ASBC).

Debido a que se pretende una evaluación profunda de la arquitectura propuesta teniendo en cuenta la incidencia de distintos atributos de calidad, así como la determinación del impacto de las decisiones arquitectónicas tomadas, se utilizará el método ATAM para la evaluación de la misma.

1.5 Líneas de Productos de Software (LPS).

Las Líneas de Productos de Software, son un conjunto de sistemas con uso intensivo de software que comparten un grupo común y regulado de características que satisfacen las necesidades de un mercado y que son desarrollados a partir de un núcleo común de activos siguiendo un método prescrito.

Este tipo de modelo de desarrollo promueve la reutilización al extremo; debido a que se reutiliza cualquier artefacto y no solo el código fuente o algún componente ejecutable.

En las Líneas de Productos de Software (LPS) se les llama activos a los elementos que se reutilizan a lo largo de la vida en la línea. Cada activo es desarrollado de forma particular, pero debe garantizarse su utilidad de forma horizontal a toda la línea. De lo contrario puede fracasar su aplicación en los productos (5).

Dentro de los activos podemos encontrar los siguientes:

- Modelos de dominio.
- Planes de desarrollo y pruebas.
- Cronogramas y presupuestos.
- Componentes ya implementados, entre otros.

Varios beneficios que se reportan con el uso de las LPS son:

- Reducción del tiempo para el mercado.
- Incremento de la productividad.
- Reducción de costos en la producción.
- Aumento de la calidad en los productos.
- Ganancia en la uniformidad de interfaces.

1.6 Desarrollo basado en componentes

Aunque el desarrollo basado en componentes no es en sí mismo un modelo de desarrollo de software, se considera importante realizar una revisión de este tema dado a su ubicuidad en las líneas de productos de software. Las LPS son, en efecto, un modelo de desarrollo basado en componentes (aunque no siempre que se desarrolla basado en componentes se está en el contexto de una línea de producto de Software).

Aunque el desarrollo basado en componentes es una disciplina que ha sido en parte sobrepasada, no es posible prescindir de sus resultados. Pero tampoco podemos obviar las dificultades que han sobrevivido como:

- No existe consenso sobre qué es y cómo especificar un componente.

- El ciclo de vida de los sistemas basados en componentes, es por lo general, separado del ciclo de vida de los componentes que lo forman. Esto crea, entre otros problemas, uno particular con el seguimiento de las versiones de un componente que hay en explotación.

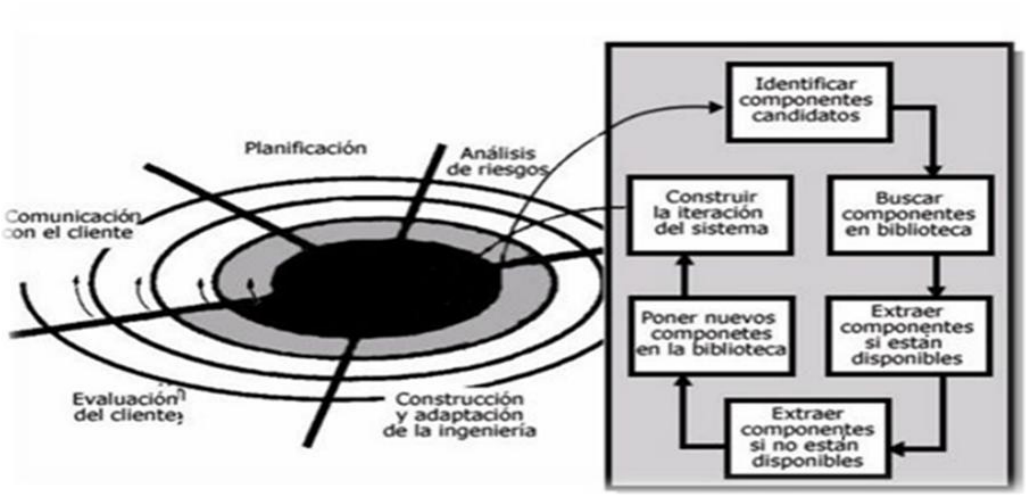


Figura # 2 Desarrollo basado en componentes

1.7 Ingeniería de Software basada en componentes.

La Ingeniería de Software basada en Componentes (Component Based Software Engineering, CBSE) está enmarcada en la Ingeniería de Software y existe un interés cada vez mayor en ella. El interés está dado primeramente por la concepción de las factorías de software que plantean la posibilidad de implantar patrones de producción industrial, al desarrollo de software y los preceptos de que es mucho más fácil y rápido reutilizar componentes ya desarrollados y probados, a tener que implementarlos desde cero.

Conclusiones del Capítulo

A partir de los objetivos propuestos para este capítulo se puede concluir lo siguiente:

- Se realizó el Diseño teórico-metodológico definitivo de la investigación, realizando así un estudio del arte de la Arquitectura de Software.
- Se analizaron las características fundamentales que debe tener el arquitecto de software.
- Se analizó además los principales parámetros que debe definir la calidad de la Arquitectura de Software propuesta.

- Se realizó un estudio detallado de los métodos existentes para la evaluación de la arquitectura, ya que estos se centran en la evaluación temprana de la misma, específicamente después de la definición.

- También se ha definido la Ingeniería de Software Basada en Componentes, y cómo la misma tiene una relación importante con la Arquitectura de Software.

.

CAPÍTULO2: TECNOLOGÍAS Y PATRONES

Introducción

Este capítulo es el resultado de la búsqueda y el análisis de la información correspondiente al proceso de diseño de la arquitectura del sistema. También se realizará un profundo análisis de los estilos arquitectónicos y patrones de arquitectura que se ajustan a la solución adecuada; además se definirá el marco de trabajo o framework que dará soporte a la aplicación, como las herramientas, metodología y tecnologías informáticas necesarias para su desarrollo y posterior explotación.

2.1 Plataforma tecnológica

Plantear un soporte al desarrollo del sistema con el uso de herramientas informáticas que faciliten la construcción del mismo y definir la plataforma tecnológica a utilizar, constituye una de las buenas prácticas de la arquitectura de software. Para precisar dicha plataforma se deben tomar un conjunto de decisiones técnicas que restringen el diseño y la implementación del sistema.

La prospección tecnológica sobre herramientas,librerías,framework, estilos, patrones y prácticas a implementar en la aplicación se convierte en la meta para alcanzar una arquitectura de trabajo sólida, basada en estándares, escalable y orientada a la implementación de buenas prácticas en el desarrollo.

Para ello se debe realizar un estudio exhaustivo de las herramientas y lenguajes que se adecuan a los requerimientos y presentan un espectro de funciones correspondientes a las necesidades. Dada la política hostil que el gobierno norteamericano mantiene sobre Cuba, el bloqueo económico que prohibe el acceso a las nuevas tecnologías de la informática y las comunicaciones, la premisa en la selección de las herramientas, metodología y tecnologías fue mantener las políticas de desarrollo del centro y el camino hacia la soberanía tecnológica mediante la utilización de software libre.

2.2 Tecnologías del lado del Servidor

Son aquellos lenguajes que son reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato comprensible para él.

Lenguaje de Programación: PHP

PHP es un lenguaje de programación interpretado, es un acrónimo recursivo que significa PHP Hypertext Pre-processor. Diseñado originalmente para la creación de páginas web dinámicas, de

propósito general ampliamente difundido, principalmente en interpretación del lado del servidor (server-side scripting) aunque puede ser incrustado dentro de código HTML. Publicado bajo la PHP License, considerada como software libre.

Paradigma: Multiparadigma.
Tipo de dato: dinámico.
Sistema operativo: Multiplataforma.
Soporte para bases de datos: MySQL, PostgreSQL, Oracle, MS SQL Server, Sybase mSQL Informix, entre otras.

Principales ventajas

- Capacidad de expandir su potencial utilizando la enorme cantidad de módulos (llamados extensiones).
- Posee una amplia documentación.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos
- Permite las técnicas de Programación Orientada a Objetos.
- Biblioteca nativa de funciones sumamente amplia e incluida.
- No requiere definición de tipos de variables.
- Tiene manejo de excepciones (desde PHP5).

Para el desarrollo de los sistemas existentes en la línea se valoraron lenguajes de programación web como: Java, Python, Ruby o Perl, pero se decidió dar la solución sobre PHP, debido a que la mayoría de las aplicaciones y servicios que se ejecutan actualmente en el centro de desarrollo DATEC son desarrolladas a través de éste excelente lenguaje de programación web.

Servidor Web Apache 2.0

El servidor Apache es desarrollado por la Apache Software Foundation, presenta entre otras características mensajes de error altamente configurables y bases de datos de autenticación.
Se seleccionó como servidor web Apache 2.0 ya que constituye una buena opción a utilizar debido a que posee ventajas que lo hacen realmente exclusivo para su selección y utilización en la línea.

Principales ventajas

- Es un producto libre.
- De código abierto para plataformas Unix (BSD, GNU/Linux), Windows y otras.
- Posee gran cantidad de extensiones para diversas tecnologías.
- Cuenta con una amplia documentación.
- Gran compatibilidad con el Lenguaje de programación PHP.

Servidor de Bases de Datos PostgreSQL 9.0

PostgreSQL es un sistema gestor de base de datos objeto relacional (Object Relational Database Manager System). PostgreSQL es un descendiente del código abierto (en inglés open source) ,además soporta gran parte del SQL estándar y muchas modernas funcionalidades como: consultas complejas, llaves foráneas, disparadores (en inglés, triggers), vistas, integridad transaccional y control de versionado concurrente (MVCC en sus siglas en ingles).En el mundo del software existen muchos gestores de base de datos como son (Oracle ,MySQL Server, SQLite, Microsoft SQL Server, PostgreSQL, entre otros).Pero de acuerdo al tipo de sistema que se va a implementar y las características del mismo se escogió PostgreSQL como gestor de base de datos, porque es un gestor que además de brindar soporte para una comunidad de desarrollo internacional, elemento este que no posee MySQL pues la SunMicrosystem no provee servicios de soporte a las versiones libres de este gestor(6). También satisface las necesidades existentes con su integridad, seguridad, viabilidad y capacidad de almacenamiento lo que lo hace una excelente herramienta para la gestión profesional de bases de datos en la línea.

2.3 Tecnologías del lado del Cliente

Los lenguajes de lado del cliente son aquellos que pueden ser directamente digeridos por el navegador y no necesitan un pre-tratamiento.

Tecnología AJAX

AJAX, acrónimo de Asynchronous Java Script And XML (Java Script asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios, mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

XHTML

HTML, siglas de **Hypertext Markup Language** (Lenguaje de Marcado de Hipertexto), es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

CSS (Cascading Style Sheets).

Las hojas de estilo es una tecnología que nos permite controlar la apariencia de una página web. En un principio, los sitos web se concentraban más en su contenido que en su presentación. Con CSS podemos especificar estilos como el tamaño, fuentes, color, espaciado entre textos y recuadros ,así como el lugar donde disponer texto e imágenes en las páginas. El uso de CSS permite cambiar el aspecto de estas páginas en cuestión de minutos y en combinación con las páginas dinámicas, puedes conseguir un sitio web realmente eficiente. Para el desarrollo de la aplicación se utiliza CCS en el diseño de las páginas cliente y fue predefinido por el proyecto.

Java Script

Es un lenguaje de programación del lado del cliente, porque es el navegador el que soporta la carga de procesamiento. Es el lenguaje de programación del lado del cliente más utilizado, debido a su compatibilidad con la mayoría de los navegadores modernos. Con Java Script se pueden crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones Java Script y ejecutarlas para realizar estos efectos e interactividades, de modo que el mayor recurso, y tal vez el único, con que cuenta este lenguaje es el propio navegador.

Se debe tener en cuenta que aunque Java script sea soportado por la mayoría de los navegadores los usuarios pueden elegir la opción de Activar/Desactivar el Java script en los mismos.

Lenguaje de Etiquetado Extensible (XML)

XML son las siglas de Extensible Markup Language o Lenguaje de Marcación Extensible. Es utilizado para el intercambio de datos entre diferentes plataformas, pero de forma general engloba un conjunto de tecnologías desarrolladas sobre este para el tratamiento de datos, siendo su principal característica la de describir la información que contiene.

JSON

Notación de Objetos de Java Script (Java Script Object Notation por sus siglas en inglés) es una notación muy simple para definir objetos en Java Script, representa una alternativa al intercambio de información respecto a XML al resultar más ligero, a ello se suma que es más fácil su interpretación al poder ser decodificado directamente en un objeto.

2.4 Marcos para el desarrollo de Aplicaciones Web o Framework.

En el mundo de las aplicaciones web existen varios framework o marcos de trabajo que facilitan y automatizan muchas funcionalidades comunes en el desarrollo de este tipo de software. Entre las principales funcionalidades que proveen se encuentran, el acceso a los datos mediante ORM (Mapeo objeto-relacional, según siglas en inglés) la implementación de un patrón arquitectónico como Modelo-Vista-Controlador, así como algunos asistentes para el diseño de interfaces de usuario de mayor calidad en menos tiempo. Existen varios marcos de trabajo para PHP como CakePHP, Zend Framework, Symfony, Prado, P4A, entre otros. Los mejores y más utilizados por los desarrolladores a nivel mundial coinciden con los siguientes: CakePHP, Zend Framework y Symfony, por lo que este último fue objeto de análisis para su evaluación como posible framework a utilizar en la solución.

Framework de desarrollo Symfony

Symfony es un completo framework diseñado para optimizar el desarrollo de las aplicaciones web mediante algunas de sus principales características. Separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es no reinventar la rueda cada vez que se crea una nueva aplicación web (7).Es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y Microsoft SQL Server, sus principales características son:

- Fácil de instalar y configurar en la mayoría de plataformas (y con la garantía de que funciona correctamente en los sistemas Windows y *nix estándares).
- Independiente del sistema gestor de bases de datos.
- Implementa Front Controller como patrón derivado del Modelo –Vista-Controlador.

- Acceso a datos a través de ORM.
- Incluye soporte para AJAX.
- Soporta la instalación de extensiones o “plugins” para añadir nuevas funcionalidades.
- URLs amigables y configurables a través de un poderoso sistema de enrutamiento.
- Sencillo de usar en la mayoría de los casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos.
- Basado en la premisa de ''convenir en vez de configurar'' en la que el desarrollador solo debe configurar aquello que no es convencional.
- Sigue la mayoría de las mejores prácticas y patrones de diseño para la web. Preparado para aplicaciones empresariales y se adapta a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Fácil de extender, lo que permite su integración con las librerías de otros fabricantes.

Este framework cuenta con una documentación muy amplia, que incluye miles de páginas en el sitio oficial, una guía gratuita traducida al español conformada por más de 400 páginas a lo que se le suma excelentes tutoriales que permiten en principio la asimilación de los conceptos principales en 24 horas.

Concluido el análisis anterior, se selecciona como framework para el desarrollo a Symfony.

Framework de presentación ExtJS

ExtJS presenta un framework de desarrollo enteramente en Java Script que permite la implementación de interfaces visuales muy potentes, ofreciendo componentes para la implementación de tablas, árboles, formularios, contenedores, etc .

Principales ventajas de Ext JS

- Código reutilizable.

- Independiente o adaptable a framework diferentes(prototype, jquery, YUI).Orientada a la programación de interfaces tipo desktop en la Web.
- Soporte comercial.
- Extensa comunidad de usuarios.

Además de existir varios framework de presentación se seleccionó ExtJS ya que permite muy buenos diseños, es intuitivo, cuenta con dos licencias una Open Source y otra comercial, además de brindar muchos servicios a la hora del trabajo con las validaciones y del manejo de los errores del lado del cliente.

Mapeo de Objetos a Bases de datos (ORM)

Propel es un ORM, lo que significa que gracias a Propel se puede trabajar con una base de datos sin utilizar instrucciones SQL. Para crear, obtener, modificar o borrar datos de la base de datos, no es necesario escribir ni una sola sentencia SQL ya que se puede trabajar directamente con objetos. Propel es el ORM clásico de Symfony. Su principal ventaja es que está completamente integrado con Symfony y que decenas de plugins sólo funcionan para Propel. También es válido resaltar que en la línea se ha ido trabajando con Doctrine ya que este es el ORM del futuro de Symfony. Debido a que su principal ventaja es el rendimiento en ejecución y la forma tan concisa en la que se pueden escribir consultas muy complejas. Sin obviar que su principal problema es que todavía no dispone del mismo nivel de integración que Propel.

Independientemente de las ventajas que proporciona Doctrine, el ORM definido para ser utilizado en la línea por el alto nivel de abstracción y fácil portabilidad es. Propel .

2.5 Metodologías de Desarrollo de Software

En la actualidad, desarrollar un software es una tarea complicada, que exige una metodología de desarrollo para simplificar el trabajo y garantizar un producto con la calidad requerida. Una metodología propone un conjunto completo de actividades necesarias para transformar los requisitos de usuario en un producto. No existe una metodología de desarrollo de software universal pues las características de cada proyecto (tiempo de duración, equipo de desarrollo, recursos) exigen que se

escoja la metodología más adecuada para favorecer el trabajo de las personas que intervienen y lograr la satisfacción del cliente con el cumplimiento de los requisitos establecidos.

Proceso Unificado Abierto (Open Unified Process, OpenUP)

Open Unified Process (OpenUP) es un proceso de desarrollo unificado que está basado en Rational Unified Process (RUP). Mantiene las mismas características de RUP pues está dirigido por casos de uso, centrado en la arquitectura y además es iterativo e incremental (8).

El ciclo de vida de un proyecto según la metodología OpenUP se divide en 4 fases fundamentales:

1. Concepción: en esta fase se pretende determinar los objetivos y establecer el alcance de proyecto.

2. Elaboración: el propósito de esta fase es establecer la línea base de la arquitectura del sistema y proporcionar una base estable para el desarrollo de la siguiente fase.

3. Construcción: esta fase tiene como propósito completar el desarrollo del sistema basado en la arquitectura definida, enfocándose en el diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo.

4. Transición: en esta fase se asegura que el software esté listo para entregarse a los usuarios.

OpenUP propone 6 flujos de Trabajo:

1. Requerimientos: en este flujo de trabajo se realizan entrevistas con el cliente para comprender el problema a resolver y se definen los requerimientos.

2. Análisis y Diseño: se realiza el diseño de los requisitos que serán después implementados.

3. Implementación: en esta disciplina se realiza la implementación del sistema basándose en el diseño realizado**.**

4. Prueba: busca los defectos a los largo del ciclo de vida.

5. Gestión del Proyecto: involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.

6. Gestión de Configuración y Cambios: describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización ,actualización y control de versiones.

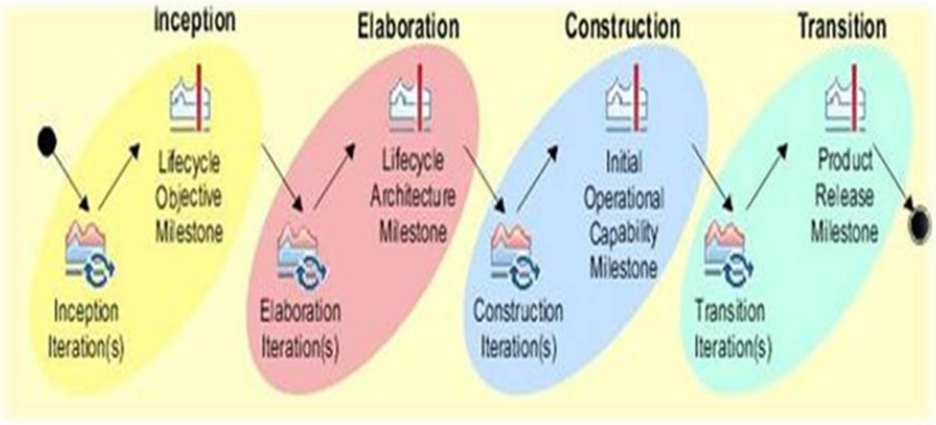


Figura # 3 Ciclo de vida de OpenUP

Actualmente en DATEC, manteniendo la filosofía de la Línea de Producción de Software (LPS) en el departamento de Integración de Soluciones se definió la utilización de la metodología OpenUP, por lo que en esta investigación se hace uso de la misma, basada en la adaptación de metodologías para el desarrollo de software siendo aplicada en cada una de las líneas de trabajo. OpenUP es una metodología ágil además de ser un proceso mínimo y suficiente, lo que significa que sólo el contenido fundamental y necesario es incluido para lograr los objetivos del proyecto.

Por lo antes expuesto y debido a que está dirigida a proyectos pequeños es considerada esta metodología como las más factible y ajustable al equipo de trabajo , para el desarrollo de la línea de productos de software Integración de Soluciones.

2.6 Ambiente de Desarrollo

Un Entorno de Desarrollo o IDE (Integrated Development Environment), es una aplicación o conjunto de estas en las que se combinan las tecnologías a utilizar para el desarrollo del software, permiten entre otras tareas: escribir el código, compilarlo o ejecutarlo, detectar errores, gestionar versiones y crear instaladores.

Eclipse

Eclipse es un entorno de desarrollo integrado (IDE) libre, fue desarrollado originalmente por IBM. Actualmente es mantenido por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios. Una de las características más importantes de Eclipse es su modularidad, pues se pueden añadir plugins según las necesidades de cada desarrollador. Una gran cantidad de estas extensiones pueden ser comúnmente encontradas en internet, tanto libres como comerciales de esta manera se puede disponer de una herramienta adaptada a las necesidades(9) .

Zend Studio for Eclipse

Zend Studio for Eclipse fue creado por Zend Technologies Inc. con el objetivo de aprovechar las bondades de Eclipse unida a las potencialidades de su propia plataforma el Zend Studio, incorpora múltiples ventajas fundamentalmente asociadas a las posibilidades de explotación de las tecnologías propias de Zend entre ellas el Zend Framework. Se distribuye comercialmente como software privativo.

NetBeans 6.9

NetBeans es un IDE que ayuda a la flexibilidad en el trabajo por sus posibilidades de modificación por parte de los desarrolladores. Ofrece además una serie de ventajas que lo hacen realmente codiciado como son: la creación de aplicaciones multiplataforma, permite que las aplicaciones creadas por el IDE se adhieran a los estándares de la industria, proporciona facilidades y garantías para la migración, facilidad de uso, cumplimiento de regulaciones y flexibilidad entre plataformas. También maneja la complejidad de la Arquitectura Orientada a Servicios (SOA) y posee soporte para PHP5.

Considerando que es un IDE libre y teniendo en cuenta que ofrece ventajas tales como un aumento de facilidades para el completamiento de código que permite agilizar el trabajo del equipo de desarrollo de la línea .Debido a esto se ha optado por NetBeans 6.9 para el desarrollo.

2.7 Lenguaje Unificado de Modelado (UML).

Un Proceso de Desarrollo de Software es la definición del conjunto de actividades que guían los esfuerzos de las personas implicadas en el proyecto a modo de plantilla y explica los pasos necesarios para terminar el proyecto . Destacar que UML no es un lenguaje de programación, es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. Su objetivo es

representar el conocimiento acerca de los sistemas que se pretenden construir y las decisiones tomadas durante su desarrollo, tanto los representados por diagramas estáticos (Casos de Uso, diagrama de clases, etc.) como los dinámicos (Diagramas de actividades, interacción, etc.) (10).

La representación en UML de un software está formado por 4+1 vistas o modelos parciales separados, relacionados entre sí, estas vistas son:

- **Vista de casos de uso.**
- **Vista lógica.**
- **Vista de implementación,**
- **Vista de procesos.**
- **Vista de despliegue.**

Por ser uno de los lenguajes más conocidos y utilizados en el mundo actualmente y además de contar con varias ventajas se seleccionó UML como Lenguaje de modelado.

2.8 Herramientas CASE

Computer Aided Software Engineering (CASE) o Ingeniería de Software Asistida por Computación, traducido al español, estas herramientas permiten organizar y manejar la información de un proyecto informático. A medida que los sistemas que hoy se construyen se tornan más y más complejos, las herramientas CASE de modelado con UML ofrecen muchos beneficios para un proyecto, logrando aplicar la metodología de análisis y diseño orientados a objetos y abstraernos del código fuente, en un nivel donde la arquitectura y el diseño se tornan más obvios y más fáciles de entender y modificar, permite también la generación de la documentación y reutilización de componentes, logrando de forma general una mayor productividad y calidad.

Visual Paradigm

Visual Paradigm es una herramienta de modelado que cuenta con una distribución gratuita, además cuenta con una interfaz gráfica muy cómoda para el usuario ayudándoles al modelado con el uso de estándares UML.

Este software cuenta además con una serie de ventajas que lo hacen ser el elegido para el modelado del presente trabajo, como son: presencia de un entorno de creación de diagramas para UML 2.0; un

diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad; utiliza un lenguaje estándar común para todo el equipo de desarrollo que facilita la comunicación; presenta capacidades de ingeniería directa e inversa; modelo y código que permanece sincronizado en todo el ciclo de desarrollo y disponibilidad de múltiples versiones para cada necesidad; además de ser multiplataforma. Se caracteriza también por ser un producto de calidad, soportar aplicaciones web y varios idiomas, generación de código para Java y exportación como HTML, fácil de instalar, actualizar y presenta compatibilidad entre ediciones.

2.9 Estilos Arquitectónicos.

Los estilos arquitectónicos, caracterizan una familia de sistemas que están relacionados por compartir propiedades estructurales y funcionales. Generalmente proveen una guía y análisis para crear una clase amplia de arquitecturas en un dominio específico donde los patrones se enfocan en solucionar los problemas más pequeños, más específicos dentro de un estilo dado. Los estilos, en cambio, expresan la arquitectura en el sentido más formal y teórico, constituyendo un tópico esencial, ellos se encuentran en el centro de la arquitectura y constituyen buena parte de su sustancia ya que un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica, debido a que el conjunto de los estilos posee las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales(11).

A continuación la clasificación de los estilos arquitectónicos.

- **Estilos de Flujo de datos.**
  - Tuberías y Filtros.
- **Estilos centrados en datos.**
  - Arquitecturas de Pizarra o repositorio.
- **Estilos de Llamada y Retorno**
  - Modelo – Vista – Controlador (MVC).
  - Arquitectura en Capas.
  - Arquitectura Orientada a Objetos.
  - Arquitectura basada en Componentes.
  - Arquitectura Cliente-Servidor.

- **Estilo de Código Móvil.**
- Arquitectura de Máquinas Virtuales.

- **Estilos Peer–To –Pee**r.
- Arquitectura basada en Eventos.
- Arquitecturas Orientas a Servicios (SOA).

A continuación se describen algunos de los estilos que se encuentran bajo la clasificación de Llamada y Retorno por su importancia en la elaboración de la solución y su utilización y seguimiento en el desarrollo de productos de software en la línea Integración de Soluciones.

Arquitectura en Capas

Es una arquitectura que ayuda a estructurar aplicaciones que pueden estar descompuestas en grupos de subtareas en las cuales cada grupo está en un nivel particular de abstracción. Este patrón describe el principio más difundido en la arquitectura. La calidad tan especial de la arquitectura de tres capas consiste en aislar la lógica de la aplicación y en convertirla en una capa intermedia bien definida y lógica del software. En la capa de presentación se realiza relativamente poco procesamiento; las ventanas envían a la capa intermedia peticiones de trabajo y este se comunica con la capa de almacenamiento del extremo posterior (11).

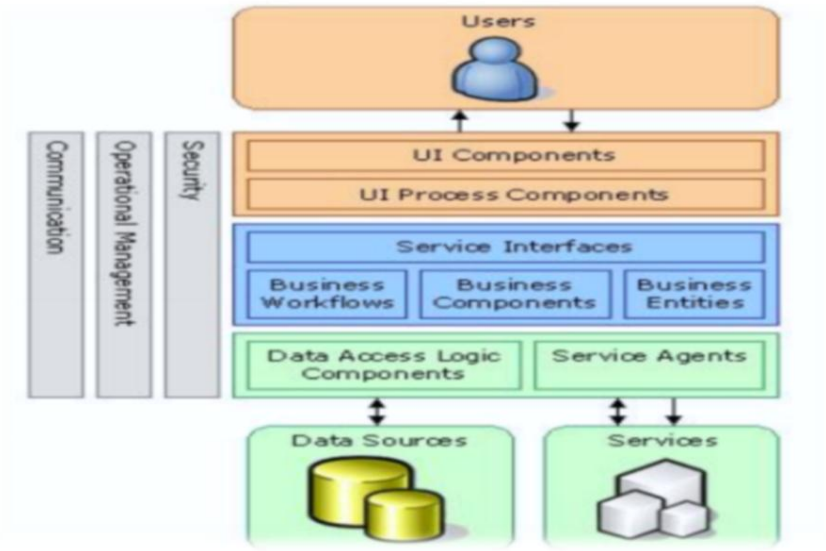
Principales ventajas

- Modularidad del sistema.
- Facilita la localización de errores.
- Mejora soporte del sistema.
- Reutilización de capas.
- Contención de cambios a una o pocas capas.

Principales desventajas

- Puede ser difícil definir que componentes ubicar en cada una de las capas.
- En ocasiones no se logra la contención del cambio y se requiere una cascada de cambios en varias capas.

- Pérdida de eficiencia.
- Dificultad de diseñar correctamente la granularidad de las capas.



Figura# 4 Arquitectura en capas

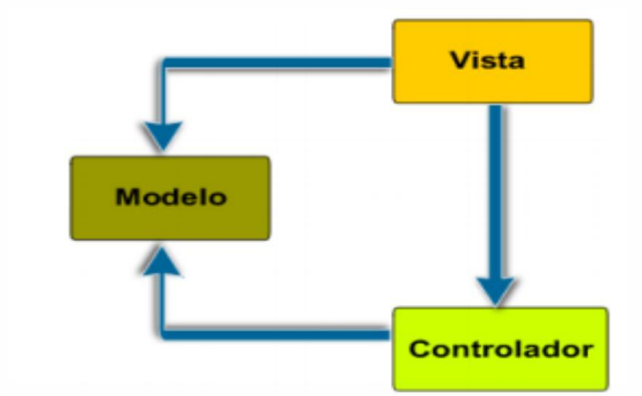
Arquitectura Modelo-Vista-Controlador (MVC)

El MVC se ve frecuentemente en aplicaciones web donde separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos:

Vista: Presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

Modelo: Esta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos. Responde a requerimientos de información sobre su estado (usualmente formulados desde la vista y responde a instrucciones y cambia el estado habitualmente desde el controlador). Es independiente de la vista y el controlador.

Controlador: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.



Figura# 5 Arquitectura Modelo-Vista –Controlador

Principales ventajas

- Separación entre interfaz, lógica de negocio y de presentación.
- Evita poner el código indebido en la capa impropia. Facilita despliegue en caso de modificaciones en modelo de datos.
- Sencillez para crear distintas representaciones de los mismos datos.

Principales desventajas

- Tener que ceñirse a una estructura predefinida puede aumentar la complejidad de la solución, hay problemas que son más difíciles de resolver respetando el patrón.
- La curva de aprendizaje para los nuevos desarrolladores se estima mayor que la de modelos más simples.
- La distribución de componentes obliga a crear y mantener un mayor número de ficheros.

Arquitectura basada en Componentes

La Arquitectura basada en componentes identifica como elemento fundamental a los componentes de software y se centra en la integración de varios de ellos para conformar un sistema, se apoya fundamentalmente de la ingeniería de software basada en componentes. Los componentes que constituyen una arquitectura pueden ejercer influencia sobre la calidad del sistema final (3).

Arquitecturas Orientadas a Objetos (AOO)

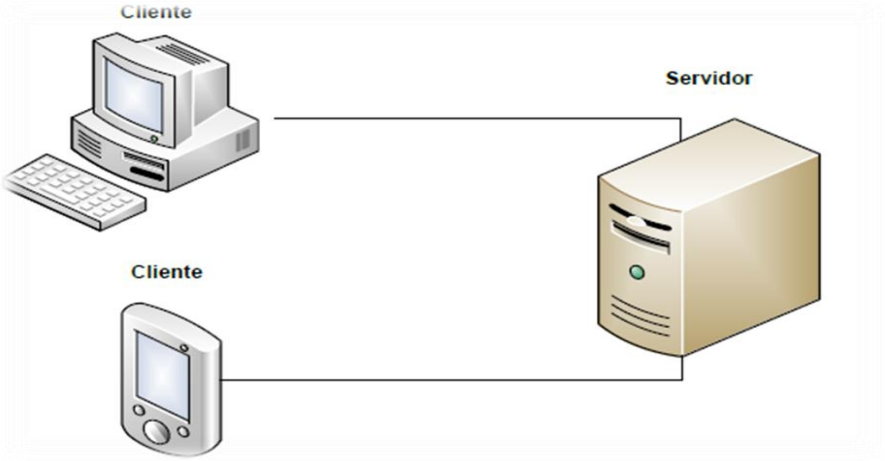
Los nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. También es importante tener presente que los objetos representan una clase de componentes llamados managers, 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 (12).

Principales ventajas: entre las cualidades la más básica concierne a que se puede modificar la implementación de un objeto sin afectar a sus clientes. De este modo es posible descomponer problemas en colecciones de agentes en interacción.

Principales desventajas :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. Los componentes se encuentran fuertemente acoplados y sólo soportan interacciones atómicas.

Arquitectura Cliente Servidor

Esta se puede definir como una arquitectura distribuida en la cual un cliente realiza peticiones a un servidor, el cual devuelve una respuesta a dichas peticiones .



Figura# 6 Arquitectura Cliente-Servidor

El Cliente normalmente es el que maneja todas las funciones relacionadas con la manipulación y despliegue de datos, como pudieran ser:

- Administrar la interfaz de usuario.
- Interactuar con el usuario.
- Procesar la lógica de la aplicación y hacer validaciones locales.
- Generar requerimientos de bases de datos.
- Recibir resultados del servidor.
- Formatear resultados.

Todas estas funciones están desarrolladas sobre plataformas que permiten construir Interfaces Gráficas de Usuario (GUI), otras de las funcionalidades de los clientes es acceder a los servicios distribuidos en cualquier parte de una red (13). El servidor es el encargado de atender las peticiones de los clientes sobre los recursos que administra. Generalmente este maneja todas las funcionalidades relacionadas con las reglas del negocio y los recursos de datos. Las características de esta arquitectura traen consigo un gran número de ventajas como son:

- Bajos costos en hardware para su implementación.

- Permite la integración de sistemas operativos de cualquier índole.
- Al presentar interfaces gráficas orientadas al cliente existe una mayor interacción con el mismo.
- Contribuye además, a proporcionar soluciones locales a las organizaciones, pero permitiendo siempre la integración de la información relevante a nivel global.

En esta arquitectura es necesario tener en cuenta la persistencia de los datos, así como la seguridad de los mismos. Esto viene dado por la compartimentación de los recursos en la red.

Es importante tener en cuenta que el uso de estos estilos arquitectónicos viene dado en dependencia de las necesidades de cada uno de los productos de software que se desarrollen en la línea, así como los patrones y tecnologías utilizadas.

2.10 Patrones

Los desarrolladores con experiencia acumulan un repertorio tanto de principios generales como de soluciones basadas en aplicar ciertos estilos que les guían en la creación de software. Estos principios y estilos si se codifican con un formato estructurado que describa el problema y la solución y se les da un nombre, podrían llamarse “Patrones”.

Por tanto, un patrón no es más que la descripción detallada de una solución adecuada a un problema concreto. El establecimiento de estos patrones comunes es lo que posibilita el aprovechamiento de la experiencia acumulada en el diseño de aplicaciones (11).

La estructura de un patrón es la siguiente:

1. Nombre: Define un vocabulario de diseño. Facilita la abstracción.
2. Problema: Describe cuando aplicar el patrón. Conjunto de fuerzas: objetivas y restricciones Prerrequisitos.
3. Solución: Elementos que constituyen el diseño (plantilla). Forma canónica para resolver fuerzas.
4. Consecuencias: Resultados. Extensiones. Consensos.

Categorías de patrones

Cada una de las categorías de patrones define un mismo nivel de abstracción o escala de aplicabilidad.

Patrones de arquitectura: Constituyen plantillas para arquitecturas de software concretas. Especifican las propiedades de la estructura global del sistema y tienen un impacto en la arquitectura de cada subsistema.

Patrones de diseño: Expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas de software. La aplicación de estos patrones no tiene un efecto en la estructura fundamental de un sistema pero pueden tener una influencia considerable en la arquitectura de un subsistema.

Idiomas: Patrones de bajo nivel, específicos para un lenguaje de programación o entorno concreto. Describe cómo implementar aspectos particulares de los componentes y sus relaciones, usando características propias del lenguaje.

Patrones de Arquitectura

En el desarrollo de proyectos de software existen muchos problemas que son comunes, como respuesta a esta situación surgen los patrones arquitectónicos. Estos patrones son la combinación problema-solución, para una situación determinada. Muchos han sido los patrones arquitectónicos que se han creado desde los inicios de la arquitectura de software. Es interesante referirse a la definición de patrones que se da en el libro Patterns of Enterprise Application Architecture de Martin Fowler, del cual se cita:

"Cada patrón describe un problema que ocurre una y otra vez en nuestro medio ambiente y, a continuación, se describe el núcleo de la solución a ese problema, de tal manera que puede utilizar esta solución un millón de veces más, sin hacerlo de la misma manera dos veces."

Los patrones de arquitectura están claramente dentro de la disciplina arquitectónica, solapándose con los estilos , aún cuando se reconoce que los patrones se refieren más bien a prácticas de reutilización y los estilos conciernen a teorías sobre la estructuras de los sistemas a veces más formales que concretas. Algunas formulaciones que describen patrones pueden leerse como si se refirieran a estilos, y también viceversa. Puede apreciarse que toda vez que surge la pregunta de qué hacer con los

estilos, de inmediato aparece una respuesta que apunta para el lado de las prácticas y los patrones. Los patrones, aún los que se han caracterizado como arquitectónicos, se encuentran más ligados al uso y más cerca del plano físico, sin disponer todavía de un lenguaje de especificación y sin estar acomodados en una taxonomía que ordene razonablemente sus clases y en un mapa que los sitúe inequívocamente en la trama de los grandes marcos de referencia (14).

Los patrones que se analizaron para la solución del sistema son los de Martin Fowler, que son patrones de arquitectura empresarial debido a que dicha arquitectura ofrece una serie de servicios a los arquitectos y desarrolladores encaminados a facilitar la implementación de aplicaciones empresariales, al tiempo que brinda una mayor cantidad de funcionalidades a los usuarios.

Algunos de los patrones que se seleccionaron son aplicados directamente en las arquitecturas diseñadas por la línea de Integración de Soluciones , y otros vienen incluidos en las tecnologías usadas como son (Symfony, ExtJS, Doctrine, Propel) .

La selección de un patrón de arquitectura o la combinación de varios es solo el primer paso cuando se diseña la arquitectura de un sistema de software.

- ***Patrón de Arquitectura Modelo-Vista-Controlador (MVC)***

La base arquitectónica del sistema en la línea de Integración de Soluciones se ha modelado haciendo uso del Patrón (MVC) con el objetivo de utilizar la separación de las responsabilidades de cada una de las capas que lo conforman y así lograr facilidades de desarrollo debido a que dicho patrón separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres partes diferentes:

- Modelo: Esta maneja todo lo relacionado con la lógica del negocio.
- Vista: Maneja la visualización de la información.
- Controlador: Estos objetos son responsables de procesar las entradas del usuario en forma de peticiones HTTP.

La vista y el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite independizar el modelo de la representación visual. En ocasiones la vista y el controlador parecen fusionados en uno solo. Esta característica facilita el cambio de las vistas o las interfaces del sistema gracias al bajo acoplamiento que existe entre el modelo y la vista (14).

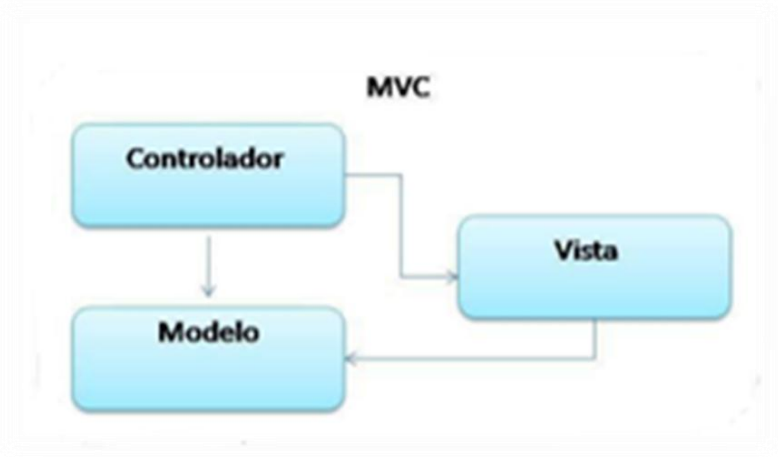


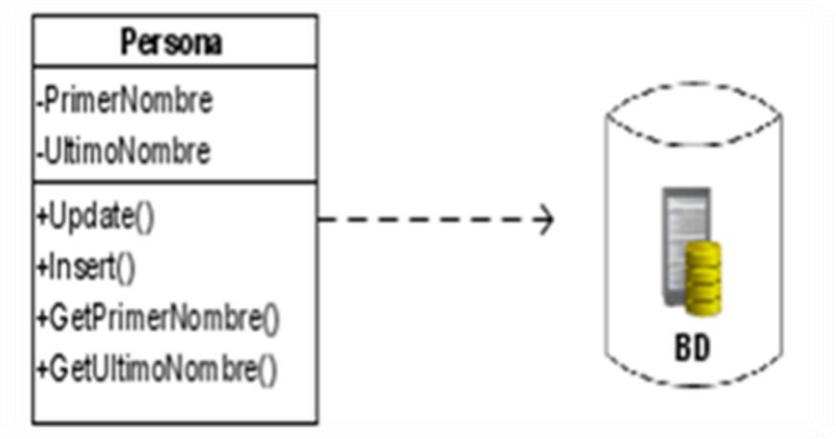
Figura # 7 Patrón Modelo Vista Controlador

El uso del patrón MVC le daría al sistema robustez y una gran escabilidad debido a que este permitiría que la lógica del negocio sea más reusable, ya que estarían separadas la presentación y la lógica del negocio.

- ***Front Controller (Controlador frontal):*** Es un patrón muy utilizado por las aplicaciones web, que describe básicamente cómo construir un sólo punto de acceso para todas las peticiones de una aplicación web. Escucha las peticiones que vienen desde una URL, y se encarga de llamar al controlador específico, posteriormente llama a la acción deseada del controlador.

- ***Template View (Vista de Plantilla):*** La idea básica de la vista de plantilla es incrustar marcadores en una página HTML cuando esta es escrita. Cuando la página se utiliza para solicitar un servicio, los marcadores se sustituirán por los resultados obtenidos de acuerdo con el servicio que brinda la página, como por ejemplo, el resultado de una consulta obtenida en una base de datos.

- ***Active Record (Registro Activo):*** Provee un objeto homólogo a una tupla de una entidad en la base de datos, encapsula el acceso a esta y la lógica de manipulación de los datos relativa a la inserción, carga, actualización y eliminación (implementado por varios framework entre ellos Propel) (14).



Figura# 8 Representación del Patrón Active Record

- ***Data Mapper (Mapeo de Datos):*** Este patrón propone definir una tabla (si se emplea una base de datos relacionales), para cada clase objeto persistente. Los atributos de los objetos que contienen tipos primitivos de datos (número, cadena, booleano y otros) se mapean en las columnas.

Si un objeto posee atributos exclusivamente de tipos primitivos de datos, el mapeo será simple. Pero la complejidad de la solución puede incrementarse un poco, ya que los objetos pueden tener atributos que se refieran a otros objetos complejos, mientras que el modelo relacional exige que los valores sean atómicos (primera forma normal) (14).

El constante cambio en los gestores de Bases de Datos es algo que golpea fuertemente los sistemas actualmente. El uso del patrón mapeo de datos, permite abstraer los componentes del gestor de bases de datos, haciendo los mismos más reutilizables. Permite además minimizar al máximo el mantenimiento de los componentes favoreciendo así la gestión de los cambios.

- ***Lazy Load (Carga Tardía***): Un objeto que no contiene todos los datos que necesitas pero sabe conseguirlo.

- ***Record Set (Colección de Registros):*** Es una representación en memoria de datos tabulares. El objetivo del patrón Record Set es representar un conjunto de líneas de base de datos relacional, con el objetivo principal de dar acceso a sus valores (una estructura de datos), a

veces con la posibilidad de la modificación (a través de una sola fila de datos de casos de puerta de enlace), a pesar del plazo fijado, las filas tienen por lo general un orden definido.

- ***Data Transfer Object:*** Objetos de datos de Transferencia (DTO), conocido anteriormente como objetos de valor o VO, Es un modelo de diseño para implementar la interfaz entre la presentación y la capa de dominio y es importante mencionar que el uso de este patrón es necesario para lograr la separación necesaria entre dichas capas, es un patrón de diseño utilizado para transferir datos entre los subsistemas de aplicación de software.

Patrones de Diseño

Los Patrones de Diseño son buenas soluciones que pueden ser aplicadas a problemas recurrentes que surgen durante el diseño de un sistema o aplicación. Para hacer sitios mejores en la Web, más funcionales y usables, es necesario romper el proceso de diseño web en pequeños pasos independientes, basados en los detalles importantes dentro de los requerimientos, con el objetivo de tener una visión de cómo se pueden aplicar los patrones en el sistema. Para los patrones que se seleccionaron se tuvieron en cuenta como marco de desarrollo o framework Symfony.

La selección de los patrones depende de la estrategia que se pretenda seguir arquitectónicamente. Para adoptar el uso de un patrón, o la combinación de varios, es recomendable que antes se realice un estudio en detalle con respecto a la solución que plantean, sus beneficios, cuándo y cómo usarlo y qué variantes se pueden usar.

Patrones GoF que se utilizan en la línea de Integración de Soluciones

El catálogo de patrones más famoso es el contenido en el libro “Design Patterns: Elements of Reusable Object-Oriented Software”, el de la banda de los 4, también conocido como el LIBRO GOF (Gang-Of-Four Book).Según el libro GOF estos patrones se clasifican según su propósito en creacionales, estructurales y de comportamiento, mientras que respecto a su ámbito se clasifican en clases y objetos.

Respecto a su propósito:

Creacionales: Ayudan a que el sistema sea independiente de cómo sus objetos son creados, compuestos o representados. El objetivo de estos patrones es abstraer el proceso de instanciación y

ocultar los detalles de cómo los objetos son creados o inicializados. Además proporcionan a los programas una mayor flexibilidad para decidir que objetos usar.

- ***Singleton (Instancia única):*** Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

- ***Builder (Constructor virtual):*** Abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto. Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.

- ***Factory Method (Método de fabricación***): Este tipo de patrón es usado frecuentemente debido a su utilidad. Su objetivo es devolver una instancia de múltiples tipos de objetos, normalmente todos estos objetos provienen de una misma clase padre mientras que se diferencian entre ellos por algún aspecto de comportamiento.

Comportamiento: Los patrones de comportamiento nos ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.

Existen otras formas de clasificar los patrones, por ejemplo en dependencia de cómo se referencian entre ellos, debido a que algunos son utilizados juntos.

Para más detalle ver Anexo 1. Clasificación de los patrones de diseño.

- ***Command (Orden):*** Tiene por propósito encapsular en un objeto la acción que satisface una petición. En el framework Symfony las acciones son un ejemplo de este patrón.

- ***Observer (Observador):*** Es un patrón de comportamiento a nivel de objeto, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él. Asume que el objeto que contiene los datos es independiente de los objetos que muestran los datos, de manera que son estos los que “observan” los cambios en dichos datos.

- ***State (Estado***): Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. De esta manera parecerá que cambia la clase del objeto.

- ***Template Method (Método plantilla):*** Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.

Patrones GRASP para asignar responsabilidades

Los patrones GRASP (Patrones de Software para la Asignación General de Responsabilidades) basan su utilidad en definir normas o principios fundamentales en el diseño orientado a objetos y las responsabilidades de estos de acuerdo a su comportamiento. Facilitan la comprensión y sistematicidad de las soluciones implementadas.

Dentro de los patrones GRASP suelen destacarse al menos cinco (Experto, Creador, Alta cohesión, Bajo acoplamiento, Controlador). Siendo los últimos 4 los más utilizados en la línea Integración de Soluciones, ya sea por el contexto en el que aparecen o las soluciones que brindan. De ahí que en la etapa de diseño se definió el uso de los principales patrones de asignación de responsabilidades con el objetivo de optimizar la implementación de la propuesta realizada.

Principales Patrones GRASP que se utilizan en la Línea de Integración de Soluciones

- **Controlador:** Asignar un controlador para evitar que las clases del negocio y las vistas tengan una comunicación directa**.**

- **Creador:** Este patrón es muy simple y su mayor beneficio es que contribuye a soportar el bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización.

- **Alta Cohesión:** Cuando se dice que una clase tiene alta cohesión, se quiere decir que el objeto tiene bien delimitadas sus responsabilidades. En la programación orientada a objetos, cada objeto tiene una responsabilidad que ha de cumplir dentro del programa, ya que la cohesión de un objeto significa cuán relacionadas están las acciones del objeto, de manera sus responsabilidades serán pocas y limitadas a su función.

- **Bajo Acoplamiento:** El bajo acoplamiento hace referencia a las relaciones que tienen los objetos entre sí dentro de un sistema. Básicamente el sistema posee bajo acoplamiento, cuando una serie de objetos tienen una relación con varios objetos, de esta manera cuando éstos últimos son cambiados, los objetos relacionados también han de verse afectados.

  Por otra parte el alto acoplamiento se da normalmente, cuando un objeto ha de saber demasiado detalles internos de otro para su funcionamiento, es decir, se rompe el encapsulamiento de otro objeto. Por ello cuando menos acoplamiento y mayor cohesión, mejor diseñado estará el sistema.

Además existen cuatro patrones GRASP adicionales que son **(Polimorfismo, Fabricación Pura, Indirección, No hables con extraños)** de estos patrones GRASP adicionales el más utilizado en la línea de productos Integración de Soluciones es:

- **Polimorfismo:** Es un patrón que nos permite asignar el mismo nombre a servicios en varios objetos, ya sea cuando los servicios se parecen o están relacionados entre sí. El uso de este patrón está acorde al espíritu del patrón Experto debido a que en vez de operar externamente sobre un objeto para autorizarlo, el objeto se autoriza a sí mismo; esto constituye la esencia de la orientación a objetos.

  Si bien es conocido que podemos caracterizar Experto como el patrón fundamental táctico, Polimorfismo será el más importante patrón estratégico en el diseño orientado a objetos.

Beneficios que brinda este patrón para su utilización en la programación orientada a objetos:

- Se añaden fácilmente las extensiones necesarias para nuevas variaciones.
- Las nuevas implementaciones se pueden introducir sin afectar a los clientes.

Conclusiones del Capítulo

A partir de los objetivos propuestos para esta Investigación se puede concluir lo siguiente:

- ✓ En el capítulo se realizó un análisis acerca de la organización del sistema.

- ✓ Se analizaron las principales tecnologías y herramientas informáticas para dar soporte al desarrollo del sistema, donde quedó definida la plataforma tecnológica para su desarrollo y posterior explotación en la línea de productos de software Integración de Soluciones.
- ✓ También se analizaron y definieron los principales patrones de arquitectura y estilos arquitectónicos que se ajustan a la solución adecuada.

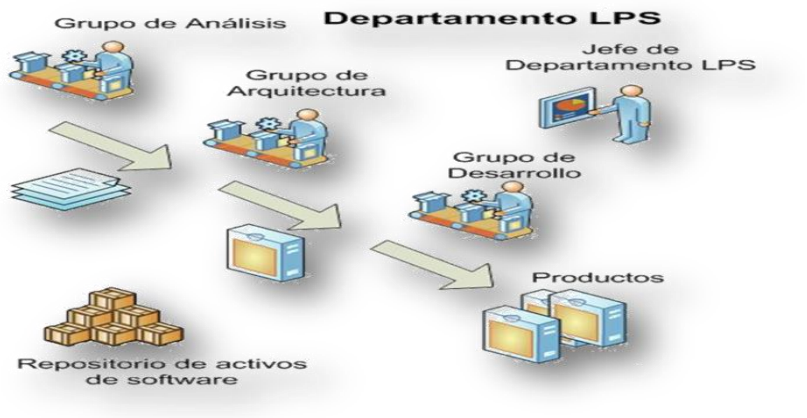
CAPÍTULO3: DEFINICIÓN DEL CATÁLOGO DE PRODUCTOS

Introducción

En este capítulo se describe la arquitectura de referencia para la línea de productos Integración de Soluciones, así como una definición conceptual de la línea misma y su estrategia de producción .Además se procede a la creación conceptual del modelo de referencia para el desarrollo de un Software y se describe cómo construir un producto a partir de los activos y de esta manera contribuir a la elaboración del catálogo de productos de la línea.

3 .1 Descripción de la línea de productos Integración de Soluciones.

En el centro de DATEC se ha propuesto la creación de líneas de productos de software para así garantizar el cumplimiento de sus objetivos estratégicos. Si bien algunas de las líneas creadas han tenido éxito y se han consolidado, otras no han terminado su formación. Una de las líneas de trabajo más destacadas debido a su gran desarrollo en este centro, es la línea de productos de software Integración de Soluciones la cual está destinada a la creación de un grupo de activos que tributen a un Paquete de Herramientas para la Ayuda a la Toma de Decisiones .Dicha línea comenzó a concebirse desde hace ya varios años. Se concibe como una suite de soluciones que permita a gerentes, empresarios, y otros tipos de ejecutivos tomar decisiones basados en la información. Incluso se proyecta la inclusión de herramientas para la sugerencia de soluciones factibles ante interrogantes en áreas temáticas como, por ejemplo, la gestión de proyectos informáticos. En el alcance de esta línea no se encuentran la codificación de modelos semánticos relacionados con áreas de conocimiento específicas que permitan a las herramientas facilitar decisiones más efectivas en las distintas áreas de trabajo; pero está diseñada de forma que puedan incorporarse estos modelos en el futuro.



Figura# 9 Estructura de un Departamento de LPS

Productos para PATDSI

El objetivo fundamental de la línea de productos PATDSI es la construcción de productos que permitan la captura de datos y su recuperación en formas factibles a la interpretación y el análisis, con el propósito fundamental de facilitar la toma de decisiones a partir de la información extraída.

Las características guías de estos productos son:

- Estos productos funcionarán tanto a escalas de instituciones pequeñas como instituciones que abarquen uno o más países. En el segundo caso se opta la replicación de datos y también por la creación de almacenes de datos.

- La consulta de la información que se está capturando se podrá monitorizar en tableros digitales que se actualizarán en tiempo real o periódicamente.

- La seguridad de los datos recogidos debe ser garantizada en todo momento.

- La interfaz de estos productos debe ser preferiblemente Web.

De forma precisa los productos de PATDSI poseen:

- Formas de captura de datos. Ya sea por entrada por formularios o consultas directas a fuentes de datos.

- Formas de análisis de datos. En particular algoritmos de análisis estadístico.

- Formas de visualización de la información con gráficos estadísticos, mapas temáticos y tableros digitales. Además de un mecanismo para la generación de reportes.

Estas tres categorías determinan en buena medida las *restricciones de los productos* en el proceso de desarrollo de activos.

Restricciones de la producción en PADTSI

En la Línea de Productos de Software Integración de Soluciones se definen varias condiciones del proceso de producción para la construcción de los productos.

En todos los casos se seguirán las siguientes restricciones:

- Todos los proyectos para la creación de los activos serán desarrollados bajo la supervisión del Grupo de Arquitectura de la Línea. Los equipos de desarrollo trabajarán de forma ágil, pero se requiere que se registren en el Redmine proyectos por cada activo y exista un repositorio de código para cada activo.

  En este caso se utilizan varios proyectos para la creación de activos, en lugar de un único proyecto ya que la idea es que, en el estado actual del centro, es manejable tener varios proyectos siguiendo un esquema de iteraciones pequeñas, aunque deben ser coordinados por el Grupo de Arquitectura de la Línea.

También existen restricciones que se aplican a un grupo concreto de activos. Los activos relacionados con las formas de visualización de la información, en particular, aquellos relacionados con la creación y visualización de reportes deben cumplir con las restricciones siguientes:

- Los activos que responden las necesidades del Generador de Reportes 2(GDR) deben estar en fase de pruebas de aceptación para el primer semestre del 2012.

- Existirá un mecanismo de migración con más de un 98% de confiabilidad para la migración de las versiones del Generador de Reportes 1 a los productos del Generador de Reportes 2.

El objetivo de la primera restricción es garantizar la supervivencia del mismo producto GDR, debido a que ya existen clientes utilizando una versión anterior de este producto. Si las próximas versiones (que estarán construidas dentro de la LPS PATDSI) se demoran demasiado es muy posible que se pierdan clientes actuales y potenciales. Los activos presentados en esta investigación preparan este producto para que sea más competitivo y tenga mayor variabilidad.

La segunda es una restricción no del proceso de producción, sino un requerimiento actual sobre este cambio en el GDR. Es evidente que los clientes que actualmente ya poseen una versión del GDR necesitan que los reportes que tengan desplazados en sus sistemas sigan funcionando sin problemas.

Estrategia para la producción en la línea

En cierto sentido se puede asegurar que los proyectos para la construcción de activos altamente relacionados con los elementos funcionales de PATDSI (ChartServer, R-Server, Generador de Formularios, etc) fueron constituidos bajo una estrategia reactiva, por ejemplo, el ChartServer se creó respondiendo a la necesidad de incluir gráficos en los reportes generados por el Generador de Reportes.

Ninguno de estos proyectos se creó de forma aislada ya que todos tienen algún grado de relación. El reto no es, por tanto, seguir una estrategia formal para la creación de los activos, sino adoptar aquella que responda apropiadamente a las necesidades que se estén presentando, ya fuese dentro de un proyecto o de la línea.

El desarrollo en la línea de productos Integración de Soluciones se enmarca en una estrategia ágil debido a que*:*

- Para cada activo se creó un proyecto.

- Los distintos proyectos productivos en la línea funcionan de forma relativamente independiente, en el sentido de que sus cronogramas, e iteraciones parciales no se correspondan en tiempo necesariamente.

- Las dependencias entre activos se manejan a nivel de línea y no por cada proyecto en particular.

  Es decir, la fecha de comienzo de un proyecto en particular se define en un portafolio general, el cual cuenta con un seguimiento constante a medida que la línea de activos vaya alcanzando una mayor madurez.

Si bien la estrategia que prima dentro de la línea es la ágil, no se descartan acciones relacionadas con otras estrategias como son: la abierta con la posible creación de proyectos comunitarios por parte de los activos tecnológicos; la automatizada al introducir técnicas de integración continua; y la estrategia de generación ,al introducir conceptos de Desarrollo Dirigido por Modelos (MDD) y la Arquitectura Dirigida por Modelos (en inglés Model Driven Architecture (MDA)) en la línea de producción.

Alcance de la línea PADTSI

PADTSI incluye un conjunto de herramientas que permiten capturar datos de distintas fuentes, consultar estos datos en forma de reportes, visualizarlos y monitorizar datos o indicadores además de programar acciones de respuesta ante estos eventos. Las aplicaciones reales que se pueden dar con este grupo de características involucran desde la captura en un formulario simple, hasta la representación en tiempo real sobre un mapa temático de indicadores geo-referenciados.

Dentro del núcleo de PATDSI se encuentran los siguientes grupos de activos:

- Componentes de captura y almacenamiento de datos.
- Componentes de consulta y visualización de información.
- Componentes tecnológicos y creados para el mismo soporte de la línea.

3.2 Inventario de activos para la construcción del Catálogo de Productos de la línea.

A continuación se enumeran todos los componentes que pueden ser utilizados para el proceso de construcción de activos. Esta lista recoge los sistemas que actualmente se desarrollan dentro de DATEC.

Este análisis fue guiado tanto por las restricciones de los productos (especialmente las categorías de activos) y también por un grupo de principios arquitectónicos que deben ser seguidos en la descripción de los mismos.

El desarrollo temporal de este análisis se realizó en varias iteraciones durante el transcurso de este trabajo, debido a que el propio desarrollo de los activos planificados modificaba la arquitectura de la línea; y ésta, a su vez, impactaba el desarrollo de los activos. En este sentido muchos activos que inicialmente se consideraban parte integral de la línea fueron declarados obsoletos y otros activos fueron definidos en su lugar. Por esta razón se podría decir que los componentes seleccionados indican los productos que son candidatos a formar parte de los activos fundamentales y esenciales de la línea de productos de software Integración de Soluciones.

Es importante tener en cuenta que en esta lista, por cada activo candidato se da una descripción necesaria para ubicarlo dentro del contexto de la línea de trabajo y así definir de forma completa el catálogo de productos existentes.

ChartServer

Descripción: Este activo es un componente que muestra datos en forma gráfica (barras, secciones, líneas, etc.) y además se puede integrar a varios productos con un costo mínimo.
Objetivo: Brindar soporte para varios tipos de gráficos.
Iteración: Ya se cuenta con una versión funcional del ChartServer 1.0.3.
Clasificación Naturaleza: Proyecto comunitario.
Clasificación Cliente: Ninguna.
Tecnología Gestor Bases Datos: Ninguna
Tecnología Lenguaje Negocio: PHP.
Tecnología IDE: NetBeans 6.9.
Tecnología Framework Desarrollo: Ninguno
Tecnología Framework Interfaz de Usuario: Ninguno
Tecnología Modelado: Visual Paradigm 6.1.
Metodología de Desarrollo: Open up.

R-Server

Descripción: Este activo es un componente que permite realizar análisis estadísticos sobre grupos de datos.
Objetivo: Realizar análisis estadísticos.
Iteración: Ya se cuenta con una versión 1.0 funcional.
Clasificación Naturaleza: Proyecto comunitario.
Clasificación Cliente: Ninguna.
Tecnología Gestor Bases Datos: Ninguna.
Tecnología Lenguaje Negocio: PHP.
Tecnología IDE: NetBeans 6.9.
Tecnología Framework Desarrollo: Lenguaje R.
Tecnología Framework Interfaz de Usuario: Ninguno.
Tecnología Modelado: Visual Paradigm 6.1.
Metodología de Desarrollo: Open up.

Generador Dinámico de Reportes

Descripción: Sistema que garantiza la generación dinámica de reportes partiendo de datos persistentes en algún origen de datos, soportado por el sistema (PostgreSQL, MySQL Server, SQLite, Microsoft SQL Server, Oracle). El sistema provee interfaces para el diseño de reportes y consultas en sentencias SQL.
Objetivo: Permitir diseñar reportes y visualizarlos.
Iteración: Terminada la versión 1.6.1 en febrero y recientemente en marzo se liberó la versión 1.7.
Clasificación Naturaleza: Proyecto de desarrollo.
Clasificación Cliente: Proyecto de Exportación.
Tecnología Gestor Bases Datos: PostgreSQL.
Tecnología Lenguaje Negocio: PHP.
Tecnología IDE: NetBeans 6.9.
Tecnología Framework Desarrollo: Symfony.
Tecnología Framework Interfaz de Usuario: EXTJS.
Tecnología Modelado: Visual Paradigm.
Metodología de Desarrollo: Open up.

Lycan Génesis

Descripción: Es un IDE para el desarrollo de aplicaciones enriquecidas para la web. Se reutiliza y configura para el desarrollo de componentes más especializados en otros dominios. Posee el respaldo institucional del Centro de Tecnologías de Gestión de Datos que lo utiliza como herramienta y como activo en el desarrollo de sus productos.

Objetivo: Facilitar el desarrollo de componentes de una manera intuitiva, rápida y cómoda.
Iteración: Para el lanzamiento de la versión 1.0.0 de Lycan Génesis falta:

- Generación de un manejador de evento.
- Permitir la edición de código de los manejadores de eventos.
- Previsualizar el componente.

Clasificación Naturaleza: Proyecto comunitario.
Clasificación Cliente: Ninguna.
Tecnología Gestor Bases Datos: SQLITE.
Tecnología Lenguaje Negocio: PHP.
Tecnología IDE: NetBeans 6.9.
Tecnología Framework Desarrollo: Symfony.
Tecnología Framework Interfaz de Usuario: EXTJS.
Tecnología Modelado: Visual Paradigm.
Metodología de Desarrollo: Open up.

SIGE

Descripción: El Sistema de Gestión Estadística tiene como objetivo principal captar la información estadística a nivel empresarial, donde a partir de formularios matriciales y formularios de encuestas se recoge dicha información.
Objetivo: Captación de información estadística a nivel empresarial,
Iteración: Ya se liberó la versión 1.7 en mayo.
Clasificación Naturaleza: Proyecto de desarrollo.
Clasificación Cliente: Ninguna.
Tecnología Gestor Bases Datos: PostgreSQL.

Tecnología Lenguaje Negocio: PHP.
Tecnología IDE: NetBeans 6.9.
Tecnología Framework Desarrollo: Symfony.
Tecnología Framework Interfaz de Usuario: EXTJS.
Tecnología Modelado: Visual Paradigm 6.1.
Metodología de Desarrollo: Open up.

Caxtor

Descripción: Es una plataforma que permite desarrollar interfaces de usuario complejas y compuestas. Caxtor incluye también un IDE de desarrollo para la creación de Interfaces de usuario. Además provee mecanismos propios de extensibilidad y composición basados en el estilo de invocación implícita. Se proyecta que en el futuro de un soporte más completo al estilo MVC cuando se incluya un DSL (del inglés *Domain-Specific Language*) que permita describir modelos de datos.
Objetivo: Desarrollar interfaces de usuario complejas y compuestas.
Iteración: Se ha terminado una versión 1.0 beta 1.
Clasificación Naturaleza: Proyecto comunitario.
Clasificación Cliente: Ninguna.
Tecnología Gestor Bases Datos: Ninguno.
Tecnología Lenguaje Negocio: PHP.
Tecnología IDE: Aptana.
Tecnología Framework Desarrollo: Ninguno.
Tecnología Framework Interfaz de Usuario: ExtJS.
Tecnología Modelado: Visual Paradigm 6.1.
Metodología de Desarrollo: Open up.

Una de las cuestiones que se debe tener en cuenta durante el desarrollo de nuevos productos es el nivel de reutilización de los activos desarrollados con anterioridad, ya que esto sería una de las variantes fundamentales para aumentar la productividad en la línea, así como una reducción del esfuerzo requerido para realizar la integración de los componentes en varios productos. La creación de este catálogo le va a permitir tanto a los arquitectos como al personal de desarrollo de los distintos equipos productivos tener un total y amplio conocimiento sobre todas las características y principales

funcionalidades que brindan los activos que se vienen desarrollando y así poder lograr la creación de un modelo de referencia a seguir para el desarrollo de productos de software y satisfacer de esta manera las necesidades existentes y objetivos trazados por la línea Integración de Soluciones.

3.3 Expediente Digital

Con el objetivo de obtener un modelo de referencia totalmente adecuado a las necesidades existentes en la línea de Integración de Soluciones se conformó un expediente digital donde para ello se realizaron algunos diagramas importantes tales como; Diagrama de Despliegue, Diagrama de Componentes y el Diagrama de Clases del Diseño, pues la realización de cada uno de estos diagramas fue de forma genérica lo que brinda la posibilidad de que puedan ser utilizados en el desarrollo de cualquier activo existente en la línea, así como la redefinición del documento de arquitectura del expediente de proyecto y de esta manera poder lograr construir una guía que sirva de apoyo a la hora de crear un producto de software determinado ,además de brindar un mejor entendimiento acerca de lo que se está desarrollando en cada equipo de trabajo.

Vista de Despliegue

La vista de despliegue describe los principales nodos físicos, ordenadores, así como los dispositivos que se necesitan para configurar una plataforma que pueda soportar la implementación del sistema. Se obtiene cuando se realiza el flujo de trabajo de Análisis y Diseño. Describe la situación de los componentes en una posible implantación del sistema de acuerdo a los requisitos iniciales. Es representada por un Diagrama de Despliegue y es un subconjunto del Modelo de Despliegue (13).

El Modelo de Despliegue provee una estructura detallada de la forma en la que los componentes se desplegarán a lo largo de la infraestructura del sistema. Detalla las capacidades de red, las especificaciones del servidor, los requisitos de hardware y otra información relacionada al despliegue del sistema propuesto. Esta vista representa el mapeo de componentes de software ejecutables con los nodos de procesamiento (hardware), tiene en cuenta los siguientes requerimientos: disponibilidad del sistema, rendimiento y escalabilidad.

Un nodo es un objeto físico que representa un recurso informático, este recurso generalmente dispone de datos persistentes y capacidad de proceso. Las conexiones entre nodos muestran las líneas de comunicación con las que el sistema tendrá que interactuar.

Figura# 10 Diagrama de Despliegue

A continuación se presenta la descripción de los nodos que conforman el diagrama de despliegue.

PC Cliente

Representa a los usuarios finales del sistema los cuales se conectan al servidor web mediante HTTP/HTTPS, utilizando un navegador web.

Servidor de Aplicaciones Web Apache2.0

Servidor web utilizado para publicar la aplicación y para lograr la conexión del sistema con la PC Cliente donde se utiliza HTTP (Hypertext Transfer Protocol) como protocolo de comunicación. Es la herramienta principal para ejecutar la lógica de negocio en el lado del servidor. Es el responsable de ejecutar el código de las páginas servidor.

Servidor de Base de Datos PostgreSQL.

Se refiere al servidor que radica en cada nodo regional donde van a estar centralizados los datos recopilados**.**

Protocolos de Comunicación

Un protocolo de comunicación es un conjunto de reglas establecidas entre dos dispositivos para permitir la comunicación entre ambos.

Conexión HTTP

Es el protocolo utilizado entre los browser de los clientes y el servidor Web. Este elemento de la arquitectura representa un tipo de comunicación no orientado a la conexión entre clientes y servidor.

TCP/IP

Es la base del Internet que sirve para enlazar computadoras. El protocolo **TCP/IP** es utilizado para establecer la conexión entre el servidor de aplicación y el servidor de base de datos.

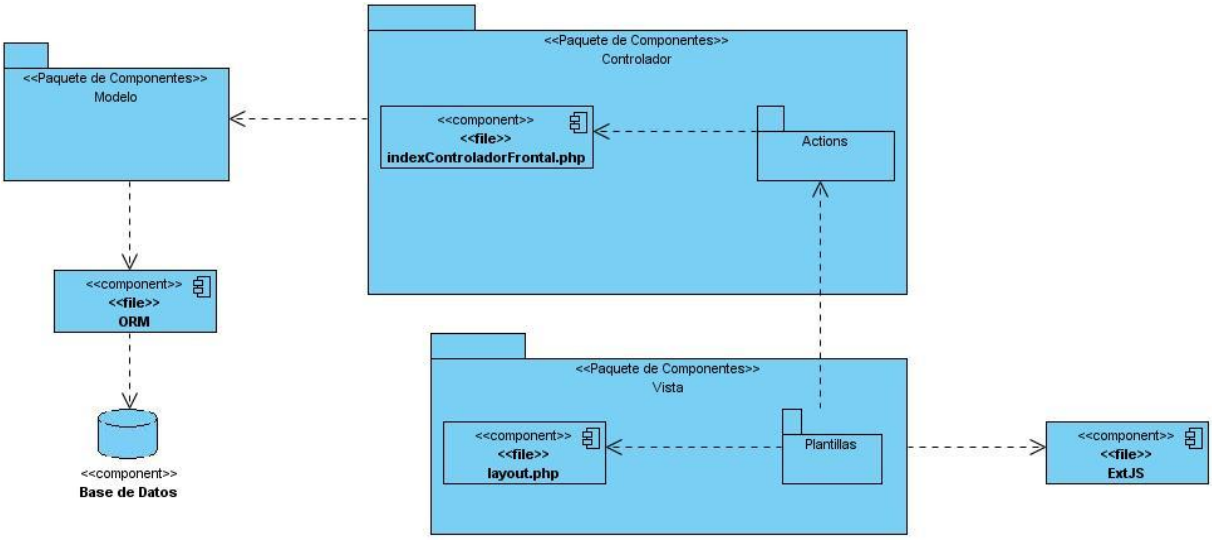
Vista de Implementación

La vista de implementación constituye una selección de los aspectos fundamentales del diagrama de componentes del sistema, el cual modela el empaquetado físico del sistema en unidades reutilizables llamadas “componentes” y sus relaciones. Un componente es una unidad física de implementación que encapsula una o más clases del diseño. Estos se pueden agrupar en subsistemas de implementación para mayor organización y claridad del diagrama.

En resumen la vista de implementación de la arquitectura muestra los elementos físicos reales más significativos del sistema. Symfony organiza el código fuente en una estructura de tipo proyecto y almacena los archivos del proyecto en una estructura estandarizada de tipo árbol (12).

Diagrama de componentes

Dentro del Modelo de Implementación se encuentran los diagramas de componentes. Un componente es el empaquetamiento físico de los elementos de un modelo, como son las clases en el modelo de diseño. El diagrama de componentes describe cómo se organizan los componentes de acuerdo con los mecanismos de estructuración disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados y cómo los componentes dependen unos de otros.



Figura# 11 Diagrama de Componentes

Se debe tener presente la utilización de Symfony como framework de desarrollo que se emplea para tomar lo mejor de la arquitectura MVC ya que este implementa de tal forma que el desarrollo de aplicaciones sea rápido y sencillo representándolo a través de tres elementos fundamentales: el modelo, la vista y el controlador.

El diagrama de componentes realizado anteriormente representa una descripción de cómo se organizan los componentes. A continuación se presenta la descripción de los paquetes y componentes que conforman el diagrama de componentes.

El paquete de componentes Controlador contiene las acciones, las cuales responden a los servicios requeridos por los casos de uso a través del **componente indexControlador Frontal.php** que es la única puerta de entrada y salida a la aplicación.

Paquete de componente Actions

Encapsula los componentes actions.class.php de cada uno de los módulos existentes en los proyectos de la línea y los implementan las clases Actions del diseño. Define las acciones que incluyen el código específico del controlador para cada página del módulo.

El paquete de componentes Modelo utiliza **ORM** para el trabajo con los datos, donde este facilita la labor de desarrollo de aplicaciones web.

Componente ORM

Para acceder a la base de datos como si fuera orientada a objetos es necesaria una interfaz que traduzca la lógica de los objetos a la lógica relacional. Esta interfaz se denomina “mapeo de objetos a bases de datos” (ORM, de sus siglas en inglés “object-relational mapping”) el cual proporciona persistencia para los objetos y un servicio de consultas.

Componente Base de Datos

Encapsula todos los datos del sistema.

En la capa Vista se encuentra reflejado el **paquete de componentes Vista**, el cual lleva incluido el archivo que representará la interfaz de usuario de acuerdo al caso de uso que se diseñe, utilizando los componentes de **ExtJS**.

Componente ExtJS

Es la capa de presentación de la aplicación, es decir con lo que interactúa el usuario.

Componente layout.php

Componente que implementa la clase layout del diseño. Contiene los elementos que se muestran de forma idéntica a lo largo de toda la aplicación.

Paquete Plantillas

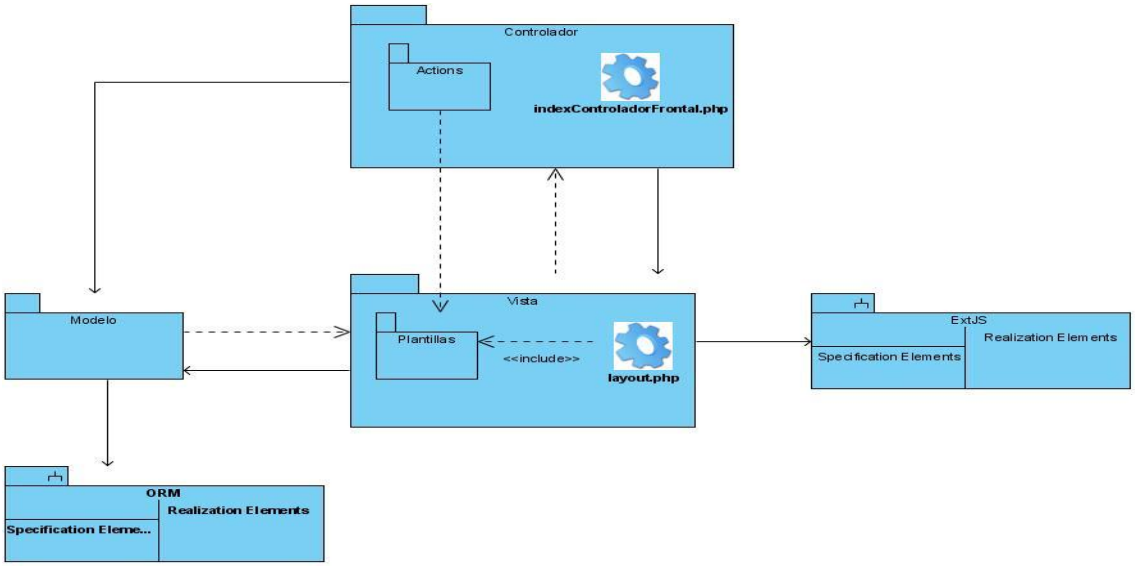
Agrupa los componentes que implementan el código correspondiente a cada plantilla que utiliza el módulo.

Vista Lógica

La vista lógica contiene las clases del diseño más importantes, organizadas por paquetes y subsistemas en capas de trabajo. Es representada por uno o varios diagramas de clases que son un subconjunto del modelo de diseño (12).

Modelo de Diseño

El Modelo de Diseño se utiliza para concebir y documentar el diseño de una aplicación. Es un modelo de objeto que describe la realización de casos de uso y sirve como una abstracción del Modelo de Implementación y del código fuente. El Modelo de Diseño se utiliza como entrada esencial para actividades en el flujo de trabajo Implementación y en el flujo de trabajo Prueba. Es un artefacto integral que abarca todas las clases de diseño, subsistemas, paquetes, colaboraciones y las relaciones entre ellos.



Figura# 12 Diagrama de Clases del Diseño

A continuación se presenta la descripción de los paquetes y subsistemas del diseño que conforman el diagrama de clases del diseño.

Paquete del Modelo

Contiene las clases que encapsulan la lógica del dominio y se encargan del acceso a los datos almacenados en el gestor de base de datos. Recoge el código para la manipulación de los datos.

Subsistema ORM: Para acceder de forma efectiva a la base de datos desde un contexto orientado a objetos es necesaria una interfaz que traduzca la lógica de objetos a la lógica relacional. Dicha interfaz se llama ORM.

Paquete Vista

Las páginas Web suelen contener elementos que se muestran de forma idéntica a lo largo de toda la aplicación: cabeceras de la página, el layout genérico, el pie de página y la navegación global. Normalmente sólo cambia el interior de la página. Por este motivo la vista se separa en un layout y en una plantilla. Habitualmente el layout es global en toda la aplicación o al menos en un grupo de páginas. Este paquete incluye las clases necesarias para presentar al usuario la lógica de la aplicación. La clase layout contiene los elementos que se muestran de forma idéntica para las páginas Web a lo largo de toda la aplicación. El **layout o plantilla** global como también se le conoce, almacena el código HTML común en todas las páginas de la aplicación para no tener que repetirlo. El contenido de la plantilla se integra en el layout o sea el layout decora la plantilla.

Paquete Plantillas

Agrupa todas las plantillas o páginas de cada módulo que se encargan de visualizar las variables definidas en el paquete del controlador. Constituyen la presentación de los datos de la acción que se están ejecutando.

Subsistema ExtJS

ExtJS es una librería construida con Java Script que proporciona una interfaz, su potencia radica en la rica colección de componentes para el diseño de GUI's del lado del cliente haciendo uso extensivo de Ajax. Entre los componentes que esta librería ofrece se encuentran cuadros de diálogo, menús, tablas editables, layout, paneles, pestañas y todo lo necesario para construir atractivos desarrollos.

Paquete Controlador

En las aplicaciones Web el controlador se encarga de realizar numerosas tareas, suele tener mucho trabajo. Una parte importante del trabajo del controlador es común para todos los controladores de la aplicación. Entre las responsabilidades comunes se encuentran el manejo de las peticiones de la aplicación, el manejo de la seguridad, cargar la configuración de la aplicación y otras tareas similares. Por este motivo el controlador normalmente se divide en un controlador frontal, que es único para cada aplicación.

La primera configuración de la aplicación se encuentra en un controlador frontal, en este caso la clase **indexControlador Frontal.php** que contiene la definición de las constantes principales, carga el archivo general de configuración y despacha la petición.

Paquete Actions

Encierra la clase Actions de cada módulo. Estas clases definen las acciones que incluyen el código específico del controlador de cada página.

Para finalizar esta investigación es importante resaltar que la arquitectura al ser el centro del desarrollo de software, debe ser descrita de tal forma que tanto desarrolladores como involucrados sean capaces de comprender la parte del sistema que les concierne, además de constituir un mapa donde se estructure desde diferentes perspectivas el software a construir.

Conclusiones del Capítulo

A partir de los objetivos propuestos para esta Investigación se puede concluir lo siguiente:

- ✓ En el capítulo se realizó una descripción acerca de cómo se encuentra estructurado actualmente PADTSI.
- ✓ También se llevó a cabo un breve análisis sobre las estrategias y restricciones que actualmente se vienen siguiendo para el desarrollo de los activos existentes.
- ✓ Se definió un catálogo de productos donde se describieron las principales características de cada activo desarrollado en la línea de productos de software Integración de Soluciones.
- ✓ Se conformó el expediente digital base para la línea y se redefinió el documento de arquitectura del expediente de proyecto.

CONCLUSIONES GENERALES

A partir de los objetivos propuestos para esta Investigación se puede concluir lo siguiente:

- ✓ Se analizaron los aspectos fundamentales sobre la disciplina Arquitectura de Software y Líneas de Productos de Software.
- ✓ Se definió la plataforma tecnológica de la línea Integración de Soluciones.
- ✓ Se definieron los estilos arquitectónicos así como los patrones de arquitectura que se ajustan a la solución adecuada.
- ✓ Se conformó el catálogo de productos a través de los activos existentes de la línea .
- ✓ Se conformó el expediente digital base para la línea y se redefinió el documento de arquitectura del expediente de proyecto.

RECOMENDACIONES

Luego de analizar los resultados del trabajo resulta factible llegar a las siguientes recomendaciones para trabajos futuros:

- Efectuar un continuo refinamiento de la arquitectura durante cada ciclo de desarrollo.
- Sugerir a la Línea de Productos de Software de Integración de Soluciones que valore el posible uso de la arquitectura propuesta y/o que consideren los principios de reutilización de componentes que esta posibilita para que puedan generalizarla.

BIBLIOGRAFÍA

1. Camacho, y otros. Introducción a la Arquitecturas de software. 2004.

2. Lobo, Robert Armando. Arquitectura de Software para el Sistema Integrado de Gestión Estadística 2.0 Nuragas. Ciudad de la Habana : s.n., 2009. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas .

3. Frometa, Jenny Infante and Hernández, Yasmany Hernández. Arquitectura de Software para el Sistema de Gestión de Reportes Dinámicos. Ciudad de la Habana : s.n., 2009. Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas .

4. Morales, Irene Suárez and Castro, Elizabeth Rodríguez. Método para evaluar la calidad de la arquitectura en cuanto a Requisitos No Funcionales en los software de Realidad Virtual. Ciudad de la Habana : s.n., 2009. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

5. Acosta, Lic. Manuel Vázquez. Definición de una arquitectura de referencia para una línea de productos de software. 2010. Tesis para optar por el grado de Master en Gestión de Proyectos Informáticos.

6. PostgreSQL, Equipo de desarrollo de. Manual de Usuario de PostgreSQL. 2005.

7. Potencier, Fabien y Zaninotto, François. Symfony. La guía definitiva.

8. Duque, Marleysi López and Ravelo, Raidel Oceguera. Herramienta en Matlab para la obtención de información de la base de datos y ficheros electroencefalograma del proyecto Mapeo Cerebral Humano Cubano. 2010. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas .

9. Foundation., The Eclipse. The Eclipse Foundation. [En línea] 10 de Diciembre de 2008. http://www.eclipse.org..

10. Jacobson, I, Booch, G y Rumbaugh, J**.** El Proceso Unifeicado de Desarrollo de Software . 84-7829-036-2.

11. Reynoso, Carlos y Kicillof, Nicolás. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. Universidad de Buenos Aires : s.n 2008.

12. Garnache, Alberto Mendoza**.** Definición de la arquitectura de software del Grupo de Desarrollo para la Gestión de Equipos Médicos. Ciudad de la Habana : s.n., Junio de 2009. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

13. Alvarez, Julio Cesar Prieto**.** Propuesta de arquitectura para el proyecto SIGESPRO. Ciudad de la Habana : s.n., Mayo del 2009.Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas .
14. Fowler, Martin, y otros**.** Patterns of Enterprise Application Architecture. s.l. : Addison Wesley, 2002. 0-321-12742-0.

15.IBM SOFTWARE GROUP. Characteristics of a Software Architect [en línea] <http://www.ibm.com/developerworks/rational/library/mar06/eeles/> [citado el 15 marzo 2006].

16. http://www.apache.org/ .

17.Documentación del SEI en la universidad Carnegie Mellon <http://www.sei.cmu.edu/publications/publications.html>.

18.1471-2000, IEEE Std. Recommended Practice for Architectural Description of Software-Intensive Systems. 2000.

19. González, Aleksander, et al. Método de Evaluación de Arquitecturas de Software Basadas en Componentes (MERCABIC). s.l. : LISI Universidad Simón Bolívar, 2007.

20. Microsoft. Estilos y Patrones en la estrategia de arquitectura, 2005.

21.Clements, Paul, Kazman, Rich and Klein, Mark. Evaluating Software Architectures: Methods and Case Studies. s.l.: Adison Wesley, 2000.

22. Garlan, David. Software Architecture: A Roadmap. En Anthony Finkelstein : The future engineering, ACM Press, 2000.

23. Bass, Len, Clement, Paul and Kazman, Rich. Software Architecture in Practice. s.l.: Addison-Wesley, 2003.

24. Sistemas Gestores de Base de Datos . Disponible en: http://www.error500.net/garbagecollector/bases_de_datos/sistema_gestor_de_base_de_dato.html (24/11/2007).

25. Kruchten, P. Architectural Blueprints—The “4+1” View Model of Software Architecture. 1995 [citado 2007 noviembre]; from:http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf.

26. WORLDWIDE INSTITUTE OF SOFTWARE ARCHITECTS. Role of the Software Architect [en línea] <http://www.wwisa.org/wwisamain/role.htm> [citado en 15 de Junio de 2008].

27. Sitio oficial del Visual Paradigm. Disponible en: http://www.visual-paradigm.com/product/vpuml. (21/01/2008).

28. BREDEMEYER. Architect Competency Framework [en línea] <http://www.bredemeyer.com/pdf_files/ArchitectCompetencyFramework.PDF> [citado en 16 de Junio de 2008]

29. http://www.bredemeyer.com/pdf_files/role.pdf .

30.Alvez, Pablo, Foti, Patricia y Scalone, Marco. 2006. Documento de Arquitectura de Software. 2006.

31.Barbacci, Mario, y otros. 2003. Using the Architecture Tradeoff Analysis Method (ATAM) to Evaluate the Software Architecture for a Product Line of Avionics Systems: A Case Study. s.l. : CMU/SEI-2003-TN-012, 2003. Technical Note.

32. Bastarrica, María Cecilia. 2003. Atributos de Calidad y Arquitectura del Software. 2003.

33. Fielding, Roy Thomas. 2000. Architectural styles and the design of network-based software architectures. University of California, Irvine : Tesis doctoral, 2000.

34.Parnas, David. 1972. On the Criteria for Decomposing Systems into Modules. . s.l. : Communications of the ACM 15(12), 1972, págs. pp. 1053-1058.

35. Propel project. 2009. Propel. [En línea] 2009. [Citado el: 6 de Mayo de 2009.] http://propel.phpdb.org/trac.

36. Wolf, Dewayne, Perry, E. y L., Alexander. Octubre de 1992. Foundations for the study of software architecture. s.l. : ACM SIGSOFT Software Engineering Notes, Octubre de 1992.

REFERENCIAS BIBLIOGRÁFICAS

1. **Camacho, y otros.** *Introducción a la Arquitecturas de software.* 2004.

2. **Lobo, Robert Armando.** *Arquitectura de Software para el Sistema Integrado de Gestión Estadística 2.0 Nuragas.* Ciudad de la Habana : s.n., 2009. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas .

3. **Frometa, Jenny Infante and Hernández, Yasmany Hernández.** *Arquitectura de Software para el Sistema de Gestión de Reportes Dinámicos.* Ciudad de la Habana : s.n., 2009. Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas .

4. **Morales, Irene Suárez and Castro, Elizabeth Rodríguez.** *Método para evaluar la calidad de la arquitectura en cuanto a Requisitos No Funcionales en los software de Realidad Virtual.* Ciudad de la Habana : s.n., 2009. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

5. **Acosta, Lic. Manuel Vázquez.** *Definición de una arquitectura de referencia para una línea de productos de software.* 2010. Tesis para optar por el grado de Master en Gestión de Proyectos Informáticos.

6. **PostgreSQL, Equipo de desarrollo de.** *Manual de Usuario de PostgreSQL.* 2005.

7. **Potencier, Fabien y Zaninotto, François.** *Symfony. La guía definitiva.*

8. **Duque, Marleysi López and Ravelo, Raidel Oceguera.** *Herramienta en Matlab para la obtención de información de la base de datos y ficheros electroencefalograma del proyecto Mapeo Cerebral Humano Cubano.* 2010. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas .

9. **Foundation., The Eclipse.** *The Eclipse Foundation.* [En línea] 10 de Diciembre de 2008. http://www.eclipse.org..

10. **Jacobson, I, Booch, G y Rumbaugh, J.** *El Proceso Unifeicado de Desarrollo de Software .* 84-7829-036-2.

11. **Reynoso, Carlos y Kicillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Universidad de Buenos Aires : s.n 2008.

12. **Garnache, Alberto Mendoza.** *Definición de la arquitectura de software del Grupo de Desarrollo para la Gestión de Equipos Médicos.* Ciudad de la Habana : s.n., Junio de 2009. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

13. **Alvarez, Julio Cesar Prieto.** *Propuesta de arquitectura para el proyecto SIGESPRO.* Ciudad de la Habana : s.n., Mayo del 2009 . Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas .

14. **Fowler, Martin, y otros.** *Patterns of Enterprise Application Architecture.* s.l. : Addison Wesley, 2002. 0-321-12742-0.

ANEXOS

		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method (107)	Adapter (139)	Interpreter (243) Template Method (325)
	Object	Abstract Factory (87) Builder (97) Prototype (117) Singleton (127)	Adapter (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Proxy (207)	Chain of Responsibility (223) Command (233) Iterator (257) Mediator (273) Memento (283) Flyweight (195) Observer (293) State (305) Strategy (315) Visitor (331)

Anexo # 1 Clasificación de los patrones de diseño

GLOSARIO

Catálogo: El catálogo muchas veces resulta ser la mejor manera y la más ordenada que tiene una empresa más a mano a la hora de presentarle al mundo los productos que fabrica o comercializa.

Entorno tecnológico: La tecnología puede definirse como el método o técnica para la conversión de recursos y resultados en el cumplimiento de una tarea específica. Por lo tanto, los métodos y técnicas no sólo se refieren al conocimiento, sino también a los medios para llevar a cabo una tarea. Es importante tener en cuenta que la innovación tecnológica se refiere al aumento en el conocimiento, la mejora de habilidades, o el descubrimiento de un nuevo medio que se extiende o mejora la capacidad de las personas para lograr una tarea dada.

IEEE: Asociación técnico profesional mundial dedicada entre otros aspectos a la estandarización, su siglas en vienen dadas por su nombre en inglés The Institute of Electrical and Electronics Engineers.

IBM: Empresa internacional mundialmente conocida por la venta de tecnología informática cuyas siglas en vienen dadas por su nombre en inglés Internacional Business Machina Corporation.

Open Source: Cualidad de algunos software de incluir el código fuente en la distribución del programa. En general se usa para referirse al software libre.

Plugins: Aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica.

Sun Microsystems: Empresa líder en servidores web y estaciones de trabajo, también desarrolladora de software. Gran contribuidor en la introducción de sistemas basados en Unix al mercado.