República de Cuba

Universidad de las Ciencias Informáticas

Facultad 2

Diseño de la arquitectura del proyecto SIGAC.

TRABAJO DE DIPLOMA

PARA OBTAR POR EL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

Autor: Vladimir Milián Núñez

Tutor: Lic. Ivannis Suarez Jerez

“Año 50 de la Revolución”

Ciudad de la Habana, Cuba. Junio del 2008

DECLARACION DE AUTORIA.

Declaro ser el único autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2008.

Vladimir Milián Núñez **Lic. Ivannis Suarez Jerez**

_____ _____

Firma del Autor Firma del Tutor

"Se debe hacer todo tan sencillo como sea posible, pero no más sencillo."

Albert Einstein

AGRADECIMIENTOS

A mi familia que siempre me apoyado en todo y brindado la fuerza para seguir.

A mis padres por ser mi guía y estar siempre a mi lado.

A Susana, mi novia, por su apoyo, amor, sonrisa y admiración.

A mis amigos, gracias por aguantarme por estos 5 años, en especial Yunior, Albin, Yadiel, Ludwig, Alfredo y los demás del piquete (ustedes saben).

A mis profesores, que ayudaron en mi formación, en especial a Abdel y Javier, guías y amigos.

A Clarita y Pablo, por su apoyo y aliento.

A Ivannis, mi tutor.

A Fidel y la Revolución, por haberme dado la posibilidad de realizar mis estudios y obtener los conocimientos necesarios para realizar este trabajo.

…y a todos los que no puse, pero que de una forma u otra han contribuido a mi formación profesional y personal, y que creyeron en mi, que no se me olvidan, pero no habría espacio, a todos, muchas gracias.

DEDICATORIA

… a mis padres a quienes nunca defraudaré.

… a Susana por su apoyo y amor.

… a la mi tío Lázaro, en paz descanse.

… a los que me aceptaron como soy.

RESUMEN.

La gestión de la información y los procesos en el Ministerio de Auditoria y Control de la República de Cuba (MAC), es un proceso de suma importancia, ya que este es un organismo rector en la lucha contra la corrupción y las ilegalidades. Actualmente, la información se maneja en su mayoría en forma manual y se almacena en formato duro (papel), lo que conlleva a gasto de recursos, demoras, inconsistencia en la información, entre otras consecuencias negativas, las cuales deben eliminarse para garantizar un control eficaz y un buen funcionamiento.

Así surge el proyecto de informatización de dicho ministerio el cual tiene como **objetivo concreto** automatizar el flujo de información y los procesos que tienen que ver con la gestión de las auditorias y controles tanto estatales como gubernamentales que son rectorados por el MAC, así como la capacitación, formación y control de los auditores del Sistema Nacional de Auditoria (SNA).

El presente trabajo de diploma tiene como objetivo el diseño de una arquitectura que permita agilizar el tiempo de desarrollo, la reutilización de componentes y el entendimiento de los demás miembros del equipo. Se plasman los resultados del estudio de las tendencias y tecnologías actuales, y de la arquitectura de software; así como la definición de las tecnologías y aspectos esenciales de diseño que posibiliten el éxito de la aplicación.

ÍNDICE.

ÍNDICE DE ILUSTRACIONES

INTRODUCCIÓN.

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

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

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

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

A partir del proceso de perfeccionamiento empresarial que se lleva en el país, las empresas trabajan para lograr una mayor eficiencia en su producción o en los servicios que brindan; como parte de este perfeccionamiento está el control y la lucha contra la corrupción.

El Ministerio de Auditoría y Control de la República de Cuba (MAC), se crea el **25 de Abril del 2001**, como un Organismo de la Administración Central del Estado encargado de: *dirigir, ejecutar y controlar la aplicación de la política del Estado y del Gobierno en materia de Auditoría Gubernamental, Fiscalización y Control Gubernamental; así como para regular, organizar, dirigir y controlar, metodológicamente, el Sistema Nacional de Auditoría (SNA). Además se encarga de la preparación y control de los auditores que perteneces a SNA, así como los talleres nacionales de Auditoria*. (1)

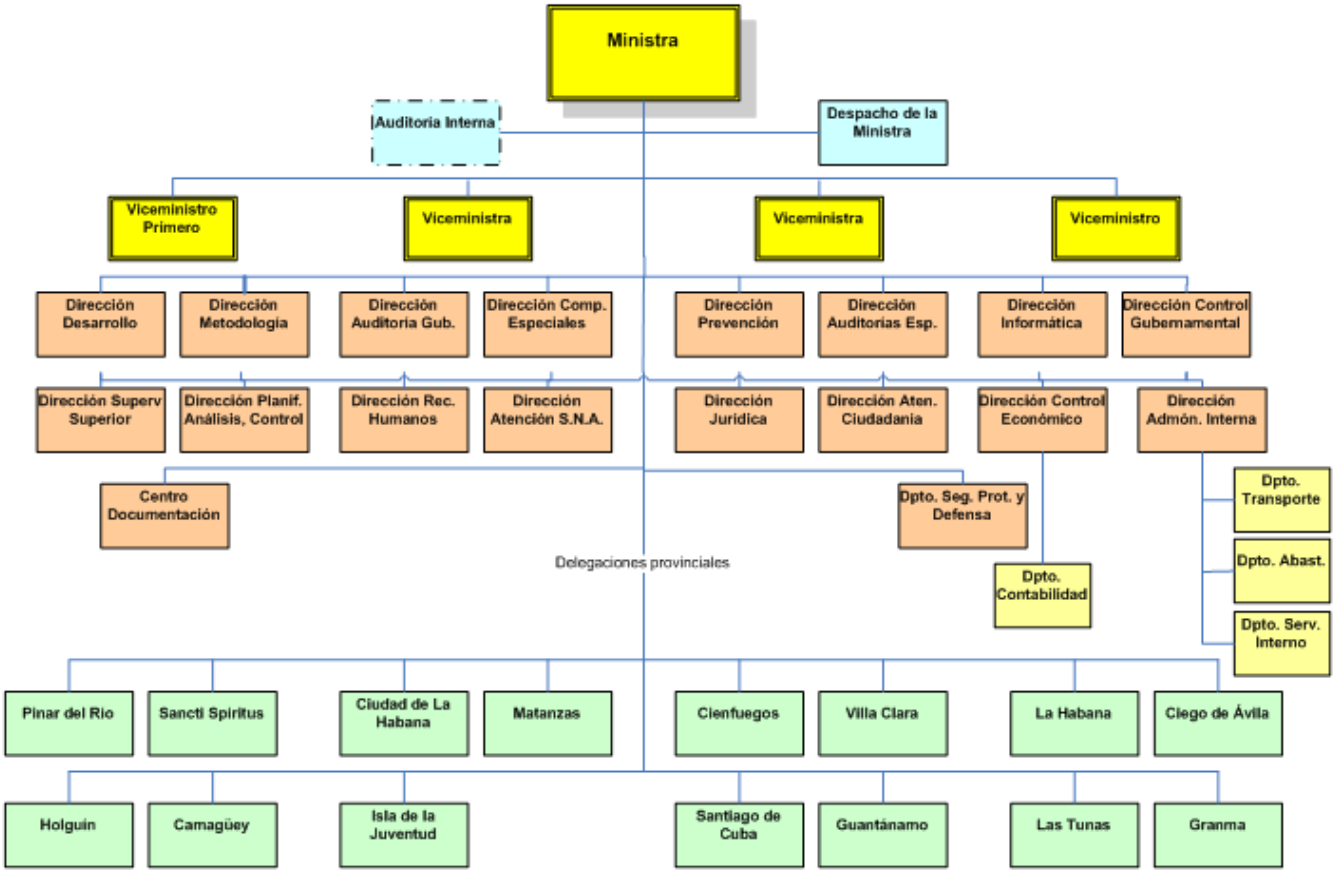


Ilustración 01. Organización y estructura del MAC

En estos momentos el MAC se ha planteado la tarea de automatizar los procesos de la entidad y el flujo de información generado por los mismos, siendo el primer ministerio del país en apostar a la completa digitalización. Las causas que conllevaron a esta decisión fueron principalmente la de brindar un servicio mas eficaz a la dirección del país y la población en general.

La aglomeración de archivos que deben ser generados (en papel) y guardados de manera organizada en locales que usan herramientas físicas (armarios), imposibilitan el buen desempeño por parte de aquellos que están al frente de dichos procesos, inhabilitando la capacidad de las estaciones de trabajo y espacios que pudieran utilizarse con un fin mas productivo. Dicha información muchas veces no es necesaria al cabo de cierto tiempo y debe ser revisada para clasificar cual se mantiene y cual es dada de baja (o se destruye).

La información digital se transmite generalmente a través de correo electrónico, guardándose en aplicaciones Microsoft Office (Word, Excel o Access) lo que provoca en muchos casos duplicidad, inconsistencia y no uniformidad de la información al no presentar dichos documentos un formato estándar (las entidades envían la información solicitada por el ministerio en plantillas creadas por las

mismas), lo que conlleva a una sobrecarga de información y por tanto, en muchos casos, mayor tiempo de procesamiento de la misma, al no contar con sistemas que ayuden en dicha tarea.

Las aplicaciones existentes en dicho ministerio, son obsoletas, con demora en el tiempo de respuesta ante las operaciones realizadas, interfaces de usuario poco amigable, no cumplen con todos los requisitos esperados y en su mayoría fueron realizadas para realizar un proceso específico y no todo el flujo de procesos de una dirección o departamento.

Las aplicaciones desarrolladas por los especialistas del MAC para suplir algunas de las necesidades de las direcciones de dicho ministerio, en su mayoría, no presentan la organización, diseño y calidad requerida, no tienen implementado sistema de seguridad y en caso de tenerlo es mínimo. Estas aplicaciones utilizan varios sistemas de bases de datos (Access, MS SQL Server, MySQL y PostgreSQL), por lo que es de primera necesidad la **integración y migración** hacia un solo gestor.

Dicho escenario plantea la necesidad de un sistema que recoja y procese toda la información que se requiere en la cede central del Ministerio y sus Delegaciones Provinciales; que permita la integración y consolidación de la documentación que tributan dichas Delegaciones, las UCAI (Unidades Centrales de Auditoria Interna) de los OACE (Organismos de la Administración Central del Estado) y los CAP (Centros de Auditorias Provinciales) al MAC; que integre y centralice toda la información en un servidor de base de datos central; que modifique e incluya (y en el peor de los escenarios, volver a realizar) los procesos informatizados existentes al nuevo sistema; que cuente con todos los mecanismos de seguridad necesarios y que pueda adaptarse a posible cambios en la estructura y organización del ministerio y sus procesos.

Para darle solución a la situación existente, fue creado el proyecto SIGAC (por las siglas del software desarrollado en el mismo: **S**istema **I**nformático de **G**estión de **A**uditoria y **C**ontrol), en el cual luego del estudio de las necesidades de procesamiento de información del MAC y la solicitud de los clientes de un sistemas que cumpla además con las características del párrafo anterior, se realizo un análisis sobre las posibilidades y beneficios de construir dicho sistema basado en tecnología web.

Las aplicaciones web ofrecen un contenido accesible para cualquier usuario desde cualquier computador; facilita la gestión eficiente de la información, fomentando la interoperabilidad en estaciones de trabajo; enriquece las aplicaciones para mejorar la usabilidad, así como los procesos de actualización del sistema se realiza solo en el servidor. Estos aspectos solo pueden alcanzarse mediante el desarrollo de elementos y procesos que forman parte de una arquitectura de software.

Vistas las ventajas que ofrece la realización de la aplicación basada en tecnologías web y el papel que juega la arquitectura para su desarrollo, se nos plantea un interesante **problema**: ¿cómo desarrollar una arquitectura que permita tomar decisiones importantes sobre los aspectos más significativos del Sistema Informático de Gestión de Auditoria y Control para tecnología web?

SIGAC será el eje principal de la infraestructura organizativa para la gestión del MAC y sus Delegaciones, por lo que el **objeto de estudio** de este trabajo, es lo referente a la gestión de los procesos y las necesidades informativas del Ministerio de Auditoria y Control. De aquí, se deriva que el **campo de acción** queda enmarcado específicamente en el diseño y descripción de la arquitectura base del proyecto SIGAC.

Con el propósito de resolver el problema antes planteado, se ha trazado como **objetivo general** de este trabajo diseñar una arquitectura de software que permita tomar decisiones importantes sobre aspectos significativos del sistema y a los desarrolladores tener una idea clara de lo que están desarrollando. De aquí se derivan objetivos más específicos como son:

- Describir la arquitectura del sistema a través de las diferentes vistas de la arquitectura según la metodología utilizada.
- Desarrollar un marco arquitectónico que permita el desarrollo de la aplicación de forma paralela en cada una de las capas lógicas.
- Determinar los estilos y patrones arquitectónicos que se emplearán durante el proceso de desarrollo y diseño del sistema.
- Definir componentes y funcionalidades reusables que agilicen el desarrollo de la aplicación.

Para darle cumplimiento a los objetivos trazados, se desarrollaron las siguientes **tareas**:

- Investigación de las diferentes propuestas arquitectónicas a partir de consultar la bibliografía especializada.
- Comparación de las distintas propuestas arquitectónicas consultadas, para obtener la más adecuada para el desarrollo de la aplicación.
- Estudio detallado de la gestión de la información y los procesos en el MAC.
- Identificación de los patrones a utilizar, fundamentación de los mismo y su aplicación.
- Definir herramientas y tecnologías a utilizar para el desarrollo.
- Declaración de los requisitos que debe cumplir el sistema.
- Creación de la documentación de la Arquitectura del Sistema.

Con el presente trabajo, se espera obtener una arquitectura para el Sistema Informático de Gestión de Auditoria y Control que sea modular, flexible, orientada a los requisitos del sistema, que sea construible, testeable y que cumpla con los parámetros de eficiencia, funcionalidad y seguridad.

El presente trabajo ha sido organizado de la siguiente manera:

Capitulo 1. Fundamentación Teórica: breve descripción de que es arquitectura; estilos y patrones arquitectónicos existentes y más usados; artefactos, metodología y herramientas a emplear en la realización de la documentación del sistema.

Capitulo 2. Características de la Arquitectura del Sistema: descripción de la arquitectura a utilizar, patrones de diseño y herramientas.

Capitulo 3. Descripción de la Arquitectura: representación de la arquitectura mediante varias de las vistas de la arquitectura (vista de casos de uso, vista lógica, vista de despliegue y vista de implementación).

Capitulo 4. Evaluación Arquitectónica: calidad de la arquitectura, atributos, métodos de evaluación de arquitecturas de software, evaluación de la arquitectura del sistema.

CAPITULO 1. FUNDAMENTACIÓN TEÓRICA.

1.1 Introducción.

En el presente capítulo se hace un análisis del surgimiento de la arquitectura de software (en lo adelante AS) y de sus tendencias, así como la descripción de los principales conceptos asociados. Se brinda además, una breve descripción de los estilos arquitectónicos existentes y más usados, la metodología, y herramientas a emplear para desarrollar la arquitectura de un sistema de software.

1.2 Arquitectura de Software.

1.2.1 Inicios de la arquitectura de software.

La AS tiene sus raíces en 1968, cuando Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera (2). Dijkstra, quien sostenía que las ciencias de la computación eran una rama aplicada de las matemáticas y sugería seguir pasos formales para descomponer problemas mayores, fue uno de los introductores de la noción de sistemas operativos organizados en capas que se comunican sólo con las capas adyacentes.

En 1975, Frederick Phillips Brooks Jr., diseñador del sistema operativo OS/360 y Premio Turing 2000, utilizaba el concepto de arquitectura del sistema para designar “la especificación completa y detallada de la interfaz de usuario” y consideraba que el arquitecto es un agente del usuario, igual que lo es quien diseña su casa (3), también distinguía entre arquitectura e implementación; mientras aquella decía qué hacer, la implementación se ocupa de cómo.

La AS siguió tomando auge con el paso de los años pero no es hasta 1992 que AS se define como disciplina de software en la publicación “Foundations for the study of software architecture” escrito por Dewayne Perry, Alexander Wolf donde planteaban (4):

“La década de 1990, creemos, será la década de la arquitectura de software…”

En 1996 Shaw y Garlan en su libro “Software Architecture. Perspectives of an Emerging Discipline” plantean:

“…más allá de las estructuras de los algoritmos y de datos del cómputo; diseñar y especificar la estructura del sistema total emerge como nueva clase de problema. Las ediciones estructurales

incluyen la organización gruesa y la estructura global del control; los protocolos para la comunicación, la sincronización, y el acceso a los datos; asignación de la funcionalidad para diseñar elementos; distribución física; composición de los elementos del diseño; escalamiento y funcionamiento; y selección entre alternativas del diseño" (5).

1.2.2 ¿Qué es la Arquitectura de Software?

Aún cuando existen cientos de enunciados sobre qué es la AS las ideas están centradas en las siguientes tres variantes:

- AS como un proceso dentro del ciclo de vida de un sistema.
- AS como la Topología del Sistema. Es decir la forma de articular los diferentes estilos y componentes dentro de una solución.
- AS como una disciplina profesional y académica.

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

"Una arquitectura es el sistema de decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y de sus interfaces por los cuales el sistema es compuesto, junto con su comportamiento según lo especificado en las colaboraciones entre estos elementos, la composición de estos elementos estructurales y del comportamiento en subsistemas progresivamente más grandes y el estilo arquitectónico que dirigen esta organización, los elementos y sus interfaces, sus colaboraciones y su composición" (6).

Este enunciado nos presenta la AS como una etapa más de la Ingeniería de Software, da una visión de la arquitectura como algo más que la construcción de herramientas o de tecnologías a usar en el desarrollo del sistema. Está enfocada a la orientación a objetos y a UML, según lo plantea la metodología de desarrollo RUP.

En el año 2000 la IEEE hace la definición "oficial" de AS en su documento IEEE 1471, que reza así:

"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".

Este documento de la IEEE 1471 ha sido adoptado por todas las personas que tienen algo que ver con la AS. Esta definición, deja claro que la AS no se inscribe en ninguna metodología de desarrollo, sino que es en sí, una disciplina que se desarrolla durante todo el ciclo de desarrollo de software.

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

"… es una descripción de los subsistemas y los componentes de un sistema informático y las relaciones entre ellos (...)" (7).

En su introducción a UML, Grady Booch, James Rumbaugh e Ivar Jacobson, específicamente refiriéndose a la metodología RUP han formulado un esquema de cinco vistas interrelacionadas que conforman la arquitectura de software. En esta perspectiva, la AS es un conjunto de decisiones significativas sobre la organización del sistema de software; la selección de elementos estructurales y sus interfaces a través de los cuales se constituye el sistema; su comportamiento (según resultados de las colaboraciones entre esos elementos); la composición de esos elementos estructurales (y comportamiento en subsistemas progresivamente mayores) y el estilo arquitectónico que guía esta organización (los elementos estáticos y dinámicos; sus interfaces, colaboraciones y composición).

En el desarrollo de una AS para la implementación de un sistema se pone de manifiesto la interacción entre los Casos de Uso (CU) y la Arquitectura; es decir los CU son directores de la arquitectura y a su vez esta guía la realización de los CU. La Arquitectura se ve condicionada por sobre qué productos de software se desea desarrollar el sistema, el o los productos de Middleware que se quieren utilizar, sistemas heredados a incorporar en el sistema, estándares y políticas corporativas, requisitos no funcionales generales y la distribución física de cada uno de los elementos del sistema.

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

Por tanto la arquitectura es:

- Vista estructural de alto nivel.
- Define estilos o combinación de estilos para una solución.
- Se concentra en los requisitos no funcionales.
- Esencial para el éxito o fracaso de un proyecto.

A continuación veamos las principales corrientes arquitectónicas:

- Arquitectura estructural, basada en un modelo estático de estilos, ADL y vistas. Constituye la corriente fundacional y clásica de la disciplina. Los representantes de esta corriente son todos académicos, mayormente de la Universidad Carnegie Mellon en Pittsburgh: Mary Shaw, Paul Clements, David Garlan, Robert Allen, Gregory Abowd, John Ockerbloom. En toda la corriente,

el diseño arquitectónico no sólo es el de más alto nivel de abstracción, sino que además no tiene por qué coincidir con la configuración explícita de las aplicaciones; rara vez se encontrarán referencias a lenguajes de programación o piezas de código.

- Arquitectura como una Etapa de ingeniería y diseño orientada a objetos. Esta corriente es una alternativa de la descrita anteriormente. Es el modelo de James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman y otros, ligado estrechamente al mundo de UML y Rational. En esta postura, la arquitectura se restringe a las fases iniciales y preliminares del proceso y concierne a los niveles más elevados de abstracción. Importa más la abundancia y el detalle de diagramas y técnicas disponibles que la simplicidad de la visión de conjunto. La definición de arquitectura que se promueve en esta corriente tiene que ver con aspectos formales a la hora del desarrollo. Las definiciones revelan que la AS, en esta perspectiva, concierne a decisiones sobre organización, selección de elementos estructurales, comportamiento, composición y estilo arquitectónico susceptibles de ser descriptas a través de las cinco vistas clásicas del modelo 4+1 de Kruchten (8) (9).
- Arquitectura basada en patrones. Esta corriente se basa principalmente en la redefinición de los estilos como patrones POSA, el diseño consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura (10) (11) (12).
- Arquitectura procesual y metodologías. Esta surge desde comienzos del siglo XXI, con centro en el SEI y con participación de algunos arquitectos de Carnegie Mellon de la primera generación y muchos nombres nuevos de la segunda: Rick Kazman, Len Bass, Paul Clements, Felix Bachmann, Fabio Peruzzi, Jeromy Carrière, Mario Barbacci y Charles Weinstock, Intenta establecer modelos de ciclo de vida y técnicas de diseño, análisis, selección de alternativas, validación, comparación, estimación de calidad y justificación económica específicas para la arquitectura de software (13).

1.2.3 Importancia y necesidad de definir una arquitectura

Si se desea construir un Software de gran complejidad como es el caso del Sistema Informático de Gestión de Auditoria y Control se necesita tener una visión común de sistemas en desarrollo. Por lo tanto se necesita una arquitectura para:

- Comprender el sistema.
- Organizar el desarrollo.
- Fomentar la reutilización.
- Hacer evolucionar el sistema.

1.3 Estilos arquitectónicos

1.3.1 ¿Qué es un estilo arquitectónico?

La En el caso de los “estilos arquitectónicos” de software son arquitecturas de software comunes, marcos de referencias arquitectónicas, formas comunes o clases de sistemas.

Los estilos de arquitectura se definen como las 4C (4):

- Componentes
- Conectores
- Configuraciones
- Restricciones (Constraints)

Estos permiten sintetizar estructuras de soluciones que luego serán refinadas a través del diseño, son pocos los estilos (alrededor de 20) que encapsulan una enorme cantidad de configuraciones concretas. Definen los patrones posibles de las aplicaciones, evitando errores arquitectónicos y permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales.

A la hora de definir un estilo arquitectónico es necesario tener en cuenta el tipo de aplicación ya que puede imponer restricciones que acotan la interacción de los componentes, además se tiene en cuenta el patrón de organización general.

Algunos de los principales estilos arquitectónicos se usan en la actualidad están divididos por Clases de Estilos las que engloban una serie de estilos arquitectónicos específicos (5):

- Estilos de Flujo de Datos
    - Tuberías y Filtros
- Estilos Centrados en Datos
    - Arquitectura de Pizarra o Repositorio
- Estilos de Llamada y Retorno
    - Modelo – Vista – Controlador (MVC)
    - Arquitectura en Capas
    - Arquitectura Orientada a Objetos
    - Arquitectura Basada en Componentes
- Estilo de Código Móvil
    - Arquitectura de Máquinas Virtuales

- Estilos Peer – To – Peer (punto a punto)
    - Arquitectura Basada en Eventos
    - Arquitectura Orientada a Servicios
    - Arquitectura Basada en Recursos.

1.3.2 Estilos de Llamada y Retorno.

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

1.3.2.1 Modelo – Vista – Controlador (MVC).

Se utiliza principalmente cuando es necesario modularizar la interfaz de usuario, las reglas de negocio y el control de eventos. Separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

Modelo: Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista y responde a instrucciones de cambiar el estado habitualmente desde el controlador). Mantiene el conocimiento del sistema. No depende de ninguna vista y de ningún controlador.

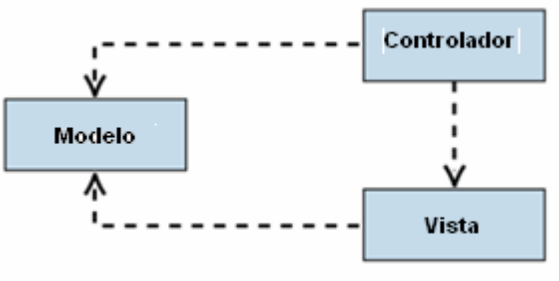


Ilustración 02. Arquitectura MVC

Vista: Maneja la visualización de la información.

Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado. Tiene tres variantes principales: Activa, Pasiva y Documento-Vista.

Ventajas:

- Se puede mostrar distintas variantes de interfaz gráfica simultáneamente.
- La interfaz tiende a cambiar más rápido que las reglas del negocio. Agregar nuevos tipos de vista no afecta el modelo.
- Evita poner el código indebido en la capa impropia. Facilita despliegue en caso de modificaciones en el modelo de datos.

Desventajas:

- Puede aumentar un poco la complejidad de la solución. Como está guiado por eventos puede ser algo más difícil de depurar.
- Si hay demasiados cambios en el modelo la vista puede provocar un constante refrescamiento de las vistas, a menos que se prevea programáticamente.

1.3.2.2 Arquitectura en Capas.

Las arquitecturas en capas constituyen uno de los estilos que aparecen con mayor frecuencia mencionados como categorías mayores del catálogo o por el contrario, como una de las posibles encarnaciones de algún estilo más envolvente. Garlan y Shaw definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.

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

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

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

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

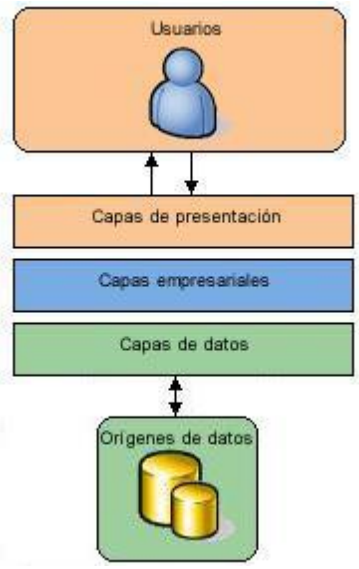


Ilustración 03. Arquitectura en capas

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

Ventajas:

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

Desventajas:

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

1.3.2.3 Arquitectura Orientada a Objetos.

Nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. En la caracterización clásica de Garlan y Shaw (14), los objetos representan una clase de componentes que ellos llaman managers (gestoras), debido a que son responsables de preservar la integridad de su propia representación.

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

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

Ventajas:

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

Desventajas:

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

Entre las limitaciones, el principal problema del estilo se manifiesta en el hecho de que para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad. La consecuencia inmediata de esta característica es que cuando se modifica un objeto (por ejemplo, se cambia el nombre de un método, o el tipo de dato de algún argumento de invocación) se deben modificar también todos los objetos que lo invocan.

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

1.3.3 Estilos Peer – To – Peer.

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

1.3.3.1 Arquitectura Orientada a Servicios.

SOA es una arquitectura de software que comienza con una definición de interfaz y construye toda la topología de la aplicación como una topología de interfaces, implementaciones y llamados a interfaces. Sería mejor llamada "*arquitectura orientada a interfaces*". SOA es una relación de servicios y consumidores de servicios, ambos suficientemente amplios para representar una función de negocios completa.

	Programación Estructurada	Objetos	Componentes	Servicios
Granularidad	Muy fina	Fina	Intermedia	Gruesa
Contrato	Definido	Privado/Público	Público	Publicado
Reusabilidad	Baja	Baja	Intermedia	Alta
Acoplamiento	Fuerte	Fuerte	Débil	Muy débil
Dependencias	Tiempo de Compilación	Tiempo de Compilación	Tiempo de Compilación	Ejecución
Ámbito de Comunicación	Intra-Aplicación	Intra-Aplicación	Intra-Aplicación	Inter-Empresas

Ilustración 04. Propiedades de la Arquitectura SOA

Dentro de los beneficios que este estilo brinda se encuentran:

Reusabilidad de Servicios:

- Disminución de tiempos y costos de desarrollo de las aplicaciones al utilizar servicios disponibles ya desarrollados, para resolver problemáticas comunes a otras aplicaciones.
- Disminución del riesgo al utilizar algo ya probado (robusto).

- Logro de mejoras globales en las aplicaciones al mejorar un servicio que está siendo usado por varias aplicaciones, agilidad frente al cambio.
- Disminución de los costos de administración y operación.

Interoperabilidad de Aplicaciones:

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

Utilización de un Medio Único de Acceso a Servicios:

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

La elección del estilo arquitectónico está dada por los escenarios que se puedan identificar de los requerimientos no funcionales. Se puede crear combinaciones de estilos arquitectónicos permitiendo lograr la satisfacción de los distintos requerimientos o escenarios.

1.4 Patrones.

Una vez que se ha decidido que estilos se van a usar en la aplicación los patrones que tienen relación con estos estilos ayudaran a derivar de esa formulación arquitectónica el diseño y ulteriormente la programación correspondiente.

1.4.1 ¿Qué son los patrones?

Los patrones fueron sugeridos por Christopher Alexander en 1977, el cual escribió una serie de libros y artículos referentes a ¿que son los patrones?, los textos más frecuentemente encontrados se refieren a patrones arquitectónicos en cuanto a arquitectura real, o sea de edificios y ciudades y no de arquitectura de aplicaciones de software.

Una de las ideas brillantes de la década de los 90 fue vincular las estructuras recurrentes, aquellas entidades que ocurrían una y otra vez; las soluciones recurrentes a problemas en determinados contextos con esta idea de Alexander que estaba referida como se dijo anteriormente a la arquitectura real. Por tanto un patron es:

- La solución a un problema en un contexto.
- Codifica conocimiento específico acumulado por la experiencia en un dominio.

Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de la solución a este problema, de tal manera que puede usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces.

1.4.2 Clasificación de los patrones.

Los patrones pueden clasificarse o dividirse en:

Patrones de Arquitectura: Relacionados a la interacción de objetos dentro o entre niveles arquitectónicos, problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, performance, modularidad, acoplamiento. Según lo expresado en el texto de Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad y Michael Stal, Pattern-Oriented Software Architecture (POSA), los patrones arquitectónicos son lo mismo que los estilos y ambos términos se usan de manera indistinta.

Patrones de Diseño: que fueron construidos dado los problemas con la claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, proponiendo como solución: Comportamiento de factoría, Clase-Responsabilidad-Contrato (CRC). En esta calificación encontramos los patrones GRAPS (General Responsability Asignment Software Patterns) que describen los principios fundamentales de la asignacion de responsabilidades a objetos y los GoF (Gans of Four) que enmarca las llamados patrones de diseño Estructurales, Creacionales y de Comportamiento.

Patrones de Análisis: Usualmente específicos de aplicación.

Patrones de Proceso o de Organización: que tratan todo lo concerniente con el desarrollo o procesos de administración de proyectos, o técnicas, o estructuras de organización.

Patrones de Idioma: que reglan la nomenclatura en la cual se escriben, se diseñan y desarrollan nuestros sistemas.

Los elementos de un patrón son:

Nombre

- Define un vocabulario de diseño.
- Facilita la abstracción.

Problema

- Describe cuando aplicar el patrón.
- Conjunto de fuerzas: objetivas y restricciones.
- Prerrequisitos.

Solución

- Elementos que constituyen el diseño (plantilla).
- Forma canónica para resolver fuerzas.

Consecuencias

- Resultados.
- Extensiones.
- Consensos.

Al contrario de los estilos arquitectónicos los patrones son muchos y a su vez muy variados y es casi imposible revisar todos los patrones que existen a la hora de hacer una determinada aplicación, por eso se recomienda el uso de los patrones que estén asociados en cada uno de los estilos que se seleccionen para el desarrollo de la arquitectura.

1.5 Tecnologías actuales a considerar

1.5.1 Metodología de desarrollo de software

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

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

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

1.5.1.1 Proceso Unificado de Desarrollo (RUP)

El Proceso Unificado de Desarrollo (RUP por sus siglas del ingles Rational Unified Process), es una metodología para el desarrollo de software orientados a objetos. Es un proceso de desarrollo de software, definido como un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software. Sin embargo, el proceso unificado es más que un proceso de trabajo, es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organizaciones y diferentes niveles de aptitud. Está constituido por 9 flujos de trabajo (los 6 primeros son conocidos como flujos de ingenieria y los 3 ultimos de apoyo): modelamiento del negocio, requisitos, análisis y diseño, implementación, prueba, instalación, administracion de configuracion y cambios, administracion de proyecto, ambiente, los cuales tienen lugar sobre 4 etapas o fases: inicio, elaboración, construcción y transición. Esta metodología es adaptable para proyectos a largo plazo y establece refinamientos sucesivos de una arquitectura ejecutable.

Características específicas de RUP:

- Dirigido por casos de uso: Esto significa que el proceso de desarrollo sigue una trayectoria que avanza a través de los flujos de trabajo generados por los casos de uso. Los casos de uso se especifican y diseñan al principio de cada iteración, y son la fuente a partir de la cual los ingenieros de prueba construyen sus casos de prueba. Estos describen la funcionalidad total del sistema.
- **Centrado en la arquitectura**: Los casos de uso guían a la arquitectura del sistema y ésta influye en la selección de los casos de uso. La arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por las plataformas de software, sistemas operativos, sistemas de gestión de bases de datos, además de otros como sistemas heredados y requerimientos no funcionales. Se representa mediante varias vistas que se centran en aspectos concretos. (15)
- Iterativo e incremental: RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y las cuales se definen según el nivel de madurez que alcanzan los productos que se van obteniendo con cada actividad ejecutada. La terminación de cada fase ocurre en el hito correspondiente a cada una, donde se evalúa que se hayan cumplido los objetivos de la fase en cuestión.

RUP junto a UML (Unified Modeling Language) constituye la metodologia estandar mas utilizada para el analisis, implementacion y documentacion de sistemas orientados a objetos.

1.5.1.2 Programación Extrema (XP)

La Programación Extrema (XP por sus siglas del ingles Xtreme Programing), es una metodología ligera de desarrollo de software que se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. La metodología consiste en una programación rápida o extrema, utilizadas para proyectos de corto plazo.

Características de XP, la metodología se basa en:

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

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

- Diseñadores y programadores se comunican efectivamente con el cliente y entre ellos mismos.
- Diseños de software sencillo y libre de complejidad o pretensiones excesivas.
- Retroalimentación de usuarios y clientes desde el primer día gracias a las baterías de pruebas.
- El software es liberado en entregas frecuentes tan pronto como sea posible.
- Los cambios se implementan rápidamente tal y como fueron sugeridos.
- Las metas en características, tiempos y costos son reajustadas, permanentemente en función del avance real obtenido.

¿Qué es lo que propone XP?

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva del proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.
- El cliente o el usuario se convierte en miembro del equipo (16).

1.5.1.3 Marco de Soluciones de Microsoft (MSF)

MSF (siglas del inglés Microsoft Solution Framework) tiene las siguientes características:

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

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

- **Modelo de Arquitectura del Proyecto**: Diseñado para acortar la planificación del ciclo de vida. Este modelo define las pautas para construir proyectos empresariales a través del lanzamiento de versiones.
- Modelo de Equipo: Este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles.
- Modelo de Proceso: Diseñado para mejorar el control del proyecto, minimizando el riesgo, y aumentar la calidad acortando el tiempo de entrega. Proporciona una estructura de pautas a seguir en el ciclo de vida del proyecto, describiendo las fases, las actividades, la liberación de versiones y explicando su relación con el Modelo de equipo.
- Modelo de Gestión del Riesgo: Diseñado para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.
- Modelo de Diseño del Proceso: Diseñado para distinguir entre los objetivos empresariales y las necesidades del usuario. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible a través de un enfoque iterativo. Las fases de diseño conceptual, lógico y

físico proveen tres perspectivas diferentes para los tres tipos de roles: los usuarios, el equipo y los desarrolladores.

- Modelo de Aplicación: Diseñado para mejorar el desarrollo, el mantenimiento y el soporte, proporciona un modelo de tres niveles para diseñar y desarrollar aplicaciones software. Los servicios utilizados en este modelo son escalables, y pueden ser usados en un solo ordenador o incluso en varios servidores.

Visión general del MSF:

Fase 1 Estrategia y alcance:

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

Fase 2 Planificación y prueba de concepto:

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

Fase 3 Estabilización:

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

Fase 4 Despliegue:

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

Se plantea que lo más importante antes de elegir la metodología que se usará para la implementación del software, es determinar el alcance que tendrá y luego determinar cuál es la que más se adapta a la aplicación.

1.5.2 Lenguajes de Descripción Arquitectónica

En la década de 1990 y en lo que va del siglo XXI, se han materializado diversas propuestas para describir y razonar en términos de arquitectura de software. Muchas de ellas han asumido la forma de lenguajes de descripción de arquitectónicas, o ADL. Estos ocupan una parte importante del trabajo arquitectónico desde la fundación de la disciplina. Los ADL permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento.

Entre las características a considerar de estos lenguajes se puede citar las siguientes:

- **Composición**: Permiten la representación del sistema como composición de una serie de partes.
- **Configuración**: La descripción de la arquitectura es independiente de la de los componentes que formen parte del sistema.
- **Abstracción**: Describen los roles o papeles abstractos que juegan los componentes dentro de la arquitectura.
- **Flexibilidad**: Permiten la definición de nuevas formas de interacción entre componentes.
- **Reutilización**: Permiten la reutilización tanto de los componentes como de la propia arquitectura.
- **Heterogeneidad**: Permiten combinar descripciones heterogéneas.
- **Análisis**: Permiten diversas formas de análisis de la arquitectura y de los sistemas desarrollados a partir de ella.

Entre los principales ADL mas usados en la actualidad están:

- ACME
- Armani
- Jacal
- CHAM

1.5.2.1 Lenguaje Unificado de Modelado (UML)

UML (siglas del ingles Unified Model Language) no es un lenguaje de programación, es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software. Un artefacto es una información que es utilizada o producida mediante un proceso de desarrollo de software.

Según diversos autores UML no es, en sí, un ADL pues su forma de expresar ciertas características, sobre todo dinámicas de las estructuras no es suficiente para los arquitectos.

UML presenta varias limitaciones como una semántica ambigua (la interpretación difiere entre stakeholders con distinta formación e intereses), adaptabilidad limitada, extensibilidad a costa del soporte de herramientas y de una especificación estándar, soporte insatisfactorio para desarrollo basado en componentes, poca orientación semántica para arquitectos (por ejemplo: cuando usar un diagrama) y falta de soporte para estilos (no se puede mostrar comunicación peer-to-peer).

Pese a las deficiencias y carencias presentadas por UML como ADL, es altamente deseable el uso del mismo por el soporte de herramientas, conocimientos, documentación, recursos, costumbre de uso y ser un lenguaje común entre todos los desarrolladores.

UML puede ser extendido con MOF (Meta Object Facility) o ser usado con extensiones OCL (Object Constraint Language), lo que convierte a UML en un ADL altamente funcional. UML 2 contiene una serie de diagramas que se utilizarían para documentar arquitecturas: diagramas de componentes y despliegue (para modelar el aspecto físico de un sistema), diagrama de implementación (para mostrar una vista estática de la configuración en tiempo de ejecución de los nodos de procesamiento y los componentes que corren en dichos nodos), diagramas de paquetes (para reflejar la organización de paquetes y como se relacionan), diagramas de estados (para mostrar como un componente cambia de estado, cuales son los estados esperados y mediante que evento se modifica), diagramas de casos de uso (para modelar los requisitos específicos o funcionales del sistema). La siguiente tabla muestra la relación entre UML y ADL:

Tabla 1. Relación entre ADL y UML

Elemento de un ADL	**Relación con UML**
Componentes	Nodos, Componentes, Clases, Interfaces, Paquetes
Conectores	Puertos, Interfaces externas, relaciones
Configuraciones o sistemas	Diagramas de componentes e implementación
Propiedades	Las dependencias se pueden ver en el diagrama de implementación o paquetes, y cada uno de los componentes tiene sus propiedades internas, aunque no exactamente como lo utilizaríamos.
Restricciones	No hay elemento UML que lo indique, salvo restricciones de dependencia.
Estilos	Esta fuera de UML, son estilos arquitectónicos
Evolución	Refinamiento de diagramas
Propiedades no funcionales	No existe elemento UML que lo represente

1.5.3 Herramientas CASE (Computer Aided Software Engineering)

"Se puede definir a una herramienta CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software." (17)

Ventajas con la utilización de las herramientas CASE:

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

Dentro de las herramientas CASE tenemos:

1.5.3.1 Rational Rose

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

1.5.3.2 Visual Paradigm

Visual Paradigm es una herramienta CASE multiplataforma que utiliza UML como lenguaje de modelado y facilita enormemente el desarrollo de software. Entre sus principales características está el

soporte a ingeniería inversa, generación de código para varios lenguajes, importación desde la herramienta Rational Rose, interoperabilidad con otras herramientas a través del intercambio de información mediante exportación/importación de XML y XMI, posee un generador de informes y editor de figuras. Tiene la capacidad de crear el esquema de clases a partir de una *base de datos* y crear la definición de base de datos a partir del esquema de clases. Está disponible en varias ediciones, cada una destinada a diferentes necesidades: Enterprise, Professional, Community, Standard, Modeler y Personal; posee una interfaz amigable y profesional, y se puede modelar en varios idiomas. Visual Paradigm se puede integrar con SVN, que es un sistema de control de versiones, permitiendo el trabajo colaborativo, lo que posibilita que múltiples usuarios trabajen sobre el mismo proyecto, genera la documentación del mismo automáticamente en varios formatos como web o pdf y se integra a Entornos de Desarrollo Integrados (IDE por sus siglas en Inglés) como Eclipse o Visual Studio. Es posible crear plantillas para las especificaciones de casos de uso y describirlos, eliminando la necesidad de utilizar una herramienta externa como editor de texto.

1.5.4 Estrategias de diseño de Arquitectura

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

Diseño arquitectónico basado en artefactos. Incluye modalidades bien conocidas de diseño orientado a objetos, tales como el OMT de Rumbaugh y el de Booch. En OMT, que puede considerarse representativo de la clase, la metodología de diseño se divide en tres fases, que son Análisis, Diseño del Sistema y Diseño de Objetos. En la fase de análisis se aplican tres técnicas de modelado que son modelado de objetos, modelado dinámico y modelado funcional. En la fase de diseño de sistema tienen especial papel lo que Rumbaugh llama implementación de control de software y la adopción de un marco de referencia arquitectónico, punto en el que se reconoce la existencia de varios prototipos que permiten ahorrar esfuerzos o se pueden tomar como puntos de partida. Algunos de esos marcos de referencia se refieren con nombres tales como transformaciones por lotes, transformaciones continuas, interfaz interactiva, simulación dinámica, sistema en tiempo real y administrador de transacciones. No cuesta mucho encontrar la analogía entre dichos marcos y los estilos arquitectónicos, concepto que en esa época todavía no había hecho su aparición. En el señalamiento de las ventajas del uso de un marco preexistente también puede verse un reflejo de la idea de patrón, una categoría que no aparece jamás en todo el marco de la OMT, aunque ya había sido propuesta por el arquitecto británico Christopher Alexander varios años antes en 1977.

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

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

Diseño arquitectónico basado en dominios. Se considera una extensión del Diseño arquitectónico basado en línea de producto, se origina en una fase de análisis de dominio que, puede ser definido como la identificación, la captura y la organización del conocimiento sobre el dominio del problema, con el objetivo de hacerlo reutilizable en la creación de nuevos sistemas. El modelo del dominio se puede representar mediante distintas formas de representación bien conocidas en ingeniería del conocimiento, tales como clases, diagramas de entidad-relación, redes semánticas y reglas. Relacionado con este paradigma se encuentra la llamada arquitectura de software específica de dominio, se puede considerar como una arquitectura de múltiples vistas, que deriva una descripción.

Diseño arquitectónico basado en patrones. Las ideas de Christopher Alexander sobre lenguajes de patrones han sido masivamente adoptadas y han conducido a la actual revolución por los patrones de diseño, sobre todo a partir del impulso que le confirieron las propuestas de la llamada “Banda de los

Cuatro". Los patrones de diseño de software propuestos por la Banda buscan codificar y hacer reutilizables un conjunto de principios a fin de diseñar aplicaciones de alta calidad. Los patrones de diseño se aplican en principio sólo en la fase de diseño, aunque a la larga se han definido y aplicado patrones en las restantes etapas de desarrollo.

Una vez terminado el analisis de las tecnologias que hacen posible el desarrollo del sistema y la comparacion entre ellas, se decidio utilizar como metodología de desarrollo a RUP, por tener entre sus caracteristicas principales: pensado para proyectos y equipos grandes, en cuanto a tamaño y duración; basado en casos de usos, es decir describe la aplicación desde el punto de vista del usuario; basado en la documentacion, gestion de riesgos, artefactos y roles; descripcion de la arquitectura mediante un conjunto de vistas (logica, de casos de usos, implementacion y despliegue principalmente); divide el desarrollo en fases y estas en iteraciones; y una constante validacion de los artefactos liberados. Se utilizara UML para documentar la arquitectura por las caracteristicas antes mencionadas, como herramienta CASE se utilizará Visual Paradigm y se realizara el diseño basado en casos de uso.

1.6 Proceso Unificado de Desarrollo y Arquitectura de Software

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

La metodología de desarrollo RUP plantea que el rol de arquitecto es el responsable por el desarrollo y mantenimiento de la arquitectura de software, la que incluye como hemos visto las principales decisiones técnicas y las restricciones sobre el diseño e implementación del proyecto.

El rol de arquitecto existe en otras metodologías de desarrollo, con actividades diferentes pero con el mismo objetivo dentro del equipo de desarrollo.

¿Qué características debe cumplir un arquitecto?

"El arquitecto ideal debe ser una persona de letras, matemático, al corriente de estudios históricos, un estudiante diligente de la filosofía, conocido de la música, no ignorante de medicina, entendido en las respuestas de consultoría jurídica, al corriente de astronomía y de cálculos astronómicos"

Resumiendo, el arquitecto del software debe tener visión, y una profundidad de la experiencia que permite tomar decisiones rápidamente y hacer el juicio adecuado, crítico en ausencia de la información completa.

Las principales actividades que realiza el arquitecto son:

- Priorizar los Casos de Uso. Definir los Casos de Uso como: críticos, secundarios, auxiliares u opcionales, lo que permite definir los módulos, subsistemas y escenarios así como la interacción entre ellos, que permita tomar decisiones hacia donde centrar los esfuerzos de implementación.
- Análisis de la arquitectura. Definir la arquitectura candidata del sistema teniendo en cuenta arquitecturas similares u otros sistemas, definir además los estilos arquitectónicos, patrones de arquitectura, los principales mecanismos de diseño arquitectónico.
- Identificar mecanismos de diseño. Refinar el análisis de la arquitectura teniendo en cuenta las restricciones impuestas por el entorno de implementación.
- Estructurar el modelo de implementación. Permite establecer la estructura en la que va a residir la implementación del sistema.
- Reutilización de elementos de diseño existentes. Permite analizar las interacciones en los diagramas de clases del Análisis para encontrar interfaces, diseño de clases y subsistemas. Refinar la arquitectura e incorporar elementos reutilizables si es posible, identificar problemas comunes a los que se pueda crear soluciones generales o comunes (patrones, o familias de productos).
- Identificar los elementos de diseño. Analizar las interacciones entre las clases de Análisis para identificar los elementos de modelo de diseño arquitectónico.
- Describir la arquitectura en tiempo de ejecución. Analizar requerimientos de concurrencia, identificar los procesos y la comunicación entre ellos, identificar el ciclo de vida de los mismos.

En este caso no se propondrá en la solución la utilización de la vista de procesos, ya que esta es opcional dependiendo de las características del sistema.

Los principales artefactos a desarrollar son:

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

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

abstracción de la implementación y es utilizada como entrada fundamental de las actividades de implementación.

Modelo de Despliegue: es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. El modelo de despliegue se utiliza como entrada fundamental en las actividades de diseño e implementación debido a que la distribución del sistema tiene una influencia principal en su diseño.

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

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

1.7 Conclusiones

En este capítulo se hizo una pequeña introducción al tema de arquitectura, se explica porque es necesario la realización de una arquitectura, se realiza el estudio de algunos de los estilos, sus características, ventajas y desventajas. Como resultado del análisis hecho, se pudo escoger las herramientas a utilizar durante el ciclo de vida completo del software, la decisión estuvo avalada por la política de uso de herramientas con soporte multiplataforma y licencias de utilización libre.

CAPITULO 2. CARACTERÍSTICAS DE LA ARQUITECTURA DEL SISTEMA

2.1 Introducción.

En el presente capítulo se describe las principales características de la arquitectura del Sistema Informático de Gestión de Auditoria y Control, los principales patrones usados en el diseño del sistema, así como de las tecnologías y herramientas de soporte al desarrollo.

2.2 Concepciones Generales.

El sistema se desarrollará a través de la metodología de desarrollo de software RUP la que define tres elementos fundamentales que conforman la guía para el desarrollo de la arquitectura como parte de la solución.

- Guiado por los Casos de Uso
- Centrado en la arquitectura
- Iterativo e incremental

Que el desarrollo de la solución sea guiado por lo casos de uso permitirá la configuración del sistema en cuanto a módulos y subsistema principalmente teniendo en cuenta la prioridad y la complejidad de los distintos casos de usos y la seguridad de que el producto final satisfaga los requerimientos reales del cliente.

Cada una de las fases en el desarrollo definidas por la metodología de desarrollo termina con la consecución de un conjunto de hitos, la terminación de cada uno de ellos no se pretende que se realicen de una vez sino que se vayan alcanzando a medida que se vayan refinando, iterando una y otra vez de acuerdo con el plan de iteraciones definidos en el Plan de Desarrollo del Sistema.

2.3 Descripción general de la arquitectura del sistema.

2.3.1 Organigrama de la arquitectura.

En el desarrollo de una aplicación de software, la identificación de un tipo de arquitectura es fundamental ya que facilitará la elaboración de un plano técnico para el diseño de la misma. Dentro de los estilos que se van utilizar en el sistema se encuentran:

- Estilo Orientado a Objetos: para la realización de la aplicación, siendo la misma una aplicación Web, fue seleccionado para su desarrollo el lenguaje de programación PHP5. Este es orientado a objetos por lo que se pueden diseñar las clases usando dicho paradigma de programación, además para modelar las diferentes clases de la aplicación Web se utilizan estereotipos, que son extensiones de UML y representan dichas clases.
- Estilo Modelo–Vista–Controlador (MVC): será utilizado como un patrón, explicándose en la sección de patrones patrones.
- Estilo Orientado a Servicios: a interacción con la dirección de RR.HH se realizara a través de web services. Dicha dirección utiliza el sistema ASSET y para manipular los datos de la misma se crearon una serie de servicios web por parte de los desarrolladores del MAC.

Estos estilos se estarán utilizando de forma combinada según sus ventajas y desempeño arquitectónico debido a que la aplicación se fundamenta en un diseño modular gestionado a través de capas, que permiten el desarrollo incremental del software, y como la lógica organizacional puede sufrir cambios (integración, creación o desaparición de procesos en el ministerio) entre mas modular y distribuido este su código, será mucho mejor para su posterior mantenimiento, por lo que la arquitectura a utilizar sería una arquitectura en capas.

- **Estilo Basado en Capas**: este modelo facilita el desarrollo del sistema debido a que en caso de algún cambio sólo se ataca al nivel requerido sin tener que revisar entre código mezclado.

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

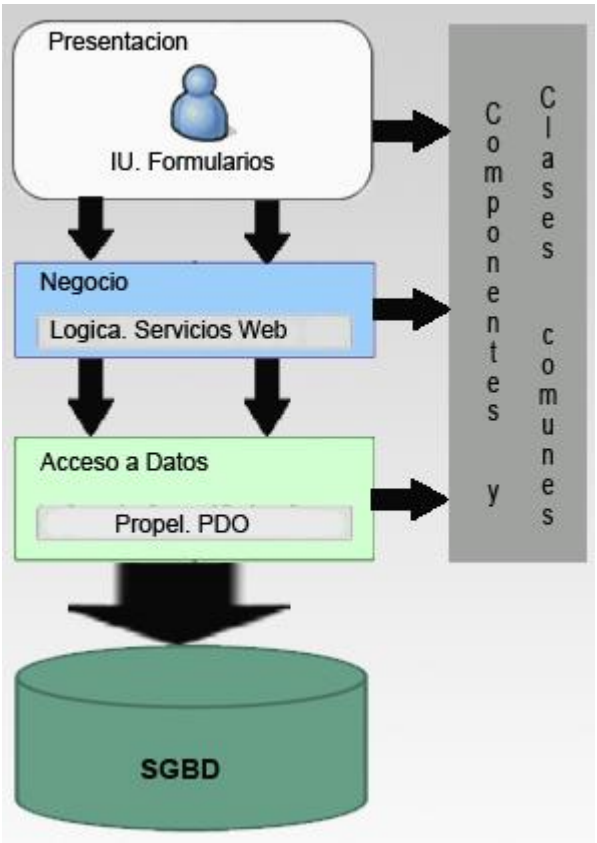


Ilustración 05. Estructura de la arquitectura de SIGAC

Presentación: Esta capa se encuentra dividida en dos subcapas fundamentales que entre las dos contienen la funcionalidad necesaria para permitir que los clientes interactúen e intercambien información con la aplicación:

- Interfaz de Usuario: La subcapa Interfaz de Usuario tiene la funcionalidad necesaria para que los usuarios intercambien información con la aplicación. Sus principales componentes son los formularios web y los componentes desarrollados a partir de la librería jquery.

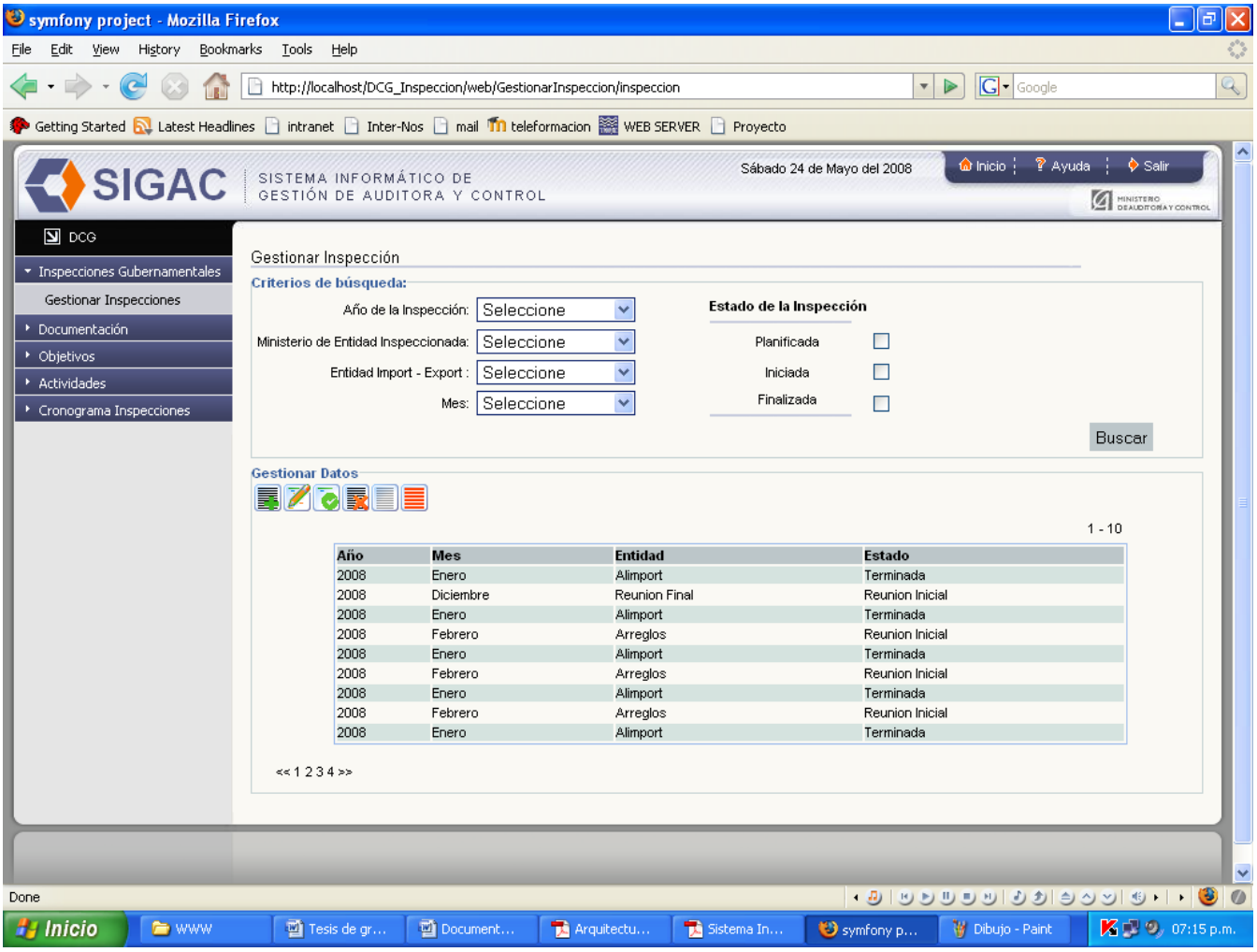


Ilustración 06. Ejemplo de Interfaz de Usuario

- Validacion: Aquí se hacen las principales validaciones de los datos a entrar en el sistema.

```
fillin:
    enabled:        on

names:
    actividad{nombre}:
        required:      Yes
        required_msg: El campo nombre es obligatorio
        validators: caracteresMaximosNombre, validarUnico

    actividad{descripcion}:
        required:      No
        validators: caracteresMaximosDescripcion

caracteresMaximosNombre:
    class:  sfStringValidator
    param:
        max: 128
        max_error: La maxima cantidad de caracteres permitida es de 128
```

Ilustración 07. Ejemplo de archivo de validación

Negocio: Esta capa almacena la lógica del negocio, en la implementación de cada una de las clases que la integran se maneja todas las transacciones de la aplicación y se aplican cada una de las reglas del negocio. En esta capa se encuentran los servicios web.

- Actions (Acciones): Es la encargada de gestionar los pedidos de la capa de presentacion y la principal relación con el “modelo” de los datos que se deben mostrar.

Acceso a Datos: Esta capa contiene la funcionalidad de acceder al repositorio de los datos, es la encargada de ejecutar la lógica de acceso a los datos; tiene dos subcapas fundamentales:

- XML Mapping (Ficheros de Mapeo) : Los ficheros de mapeo, son ficheros XML que contienen la información de la Base de Datos a la que hace referencia, contiene los campos de cada una de las tablas y sus relaciones.
- Objetos de Negocio, Base Object : Los Objetos de Negocio que se generan a partir del mapeo de las clases de las tablas de la base de datos son clases entidades de PHP que contiene los atributos y métodos para acceder a ellos, estos constituyen los objetos con que se trabaja en la aplicación.

La principal restricción que se le impone a los componentes de este estilo arquitectónico es que los componentes que pertenecen a las capas superiores solo pueden hacer uso de los componentes de sus capas inmediatas inferiores.

La organización de los componentes en cada una de las capas es la principal función del estilo de arquitectura en Capas.

2.3.2 Marco de Trabajo del Sistema.

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

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

A partir de la organización estructural del sistema en Capas se propone la utilización de dos frameworks fundamentales, distribuidos en cada capa de la aplicación con funcionalidades bien definidas y componentes implementados que permiten desarrollar nuestros propios componentes para cada capa de la aplicación:

- jQuery
- Symfony

2.3.2.1 jQuery

jQuery es una "*librería Javascript muy rápida y muy ligera que simplifica el desarrollo de la parte de cliente de las aplicaciones web*" (18). jQuery es un nuevo tipo de bibliotecas de Javascript que permite simplificar la manera de interactuar con los documentos HTML, permitiendo manejar eventos, desarrollar animaciones, y agregar interacción con la tecnología AJAX a nuestras páginas web.

Características.

- DOM interactividad y modificaciones, incluyendo soporte para CSS 1, 2 y 3 y XPath
- Eventos
- Efectos y Animaciones
- Ajax
- Extensibilidad

Utilidad.

jQuery nos provee de una serie de funcionalidades que hacen la vida del programador mucho más sencilla entre ellos podemos encontrar:

- Selectores: permiten olvidarse de document.getElementById() y del código complejo. Utiliza lo mejor de CSS 1, CSS 2, CSS 3 y XPath para seleccionar de forma sencilla cualquier elemento de la página sin necesidad de saturar el código XHTML con atributos "id". Por ejemplo, `$("div:visible")`, selecciona todos los div que no estén ocultos o `$("input[@type=radio][@checked]")`, selecciona todos los radio buttons que han sido seleccionados.
- Eventos: jQuery dispone de tantas funciones como eventos estándar de Javascript. El nombre de cada función es el mismo que el del evento, sin el habitual prefijo "on": focus(), blur(), keyup(), mouseover(), mouseup(), resize(), submit(), etc. jQuery añade un evento que no existe en Javascript: toggle(). A este método se le pasan dos funciones, cuya ejecución se alterna en función de las veces que se pincha sobre el elemento. Las veces impares se ejecuta la primera función y las veces pares se ejecuta la segunda función
- Interacciones Ajax: permite incluirlas sin ningún tipo de dificultad. Incluye muchas funciones para construir peticiones Ajax complejas y para notificar al usuario el inicio/ejecución/finalización de las mismas.
- Efectos visuales: incluye los efectos visuales más utilizados por los diseñadores (hide, show, animate, stop, slideup, slidedown, fadein, fadedown, etc), pudiendo controlar mediante sus opciones la duración de cada efecto lo

2.3.2.2 Symfony

Symfony es un completo framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Para empezar, 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 web 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 que no se debe reinventar la rueda cada vez que se crea una nueva aplicación web.

Symfony se diseñó para que se ajustara a los siguientes requisitos (19):

- 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

- Sencillo de usar en la mayoría de 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 *mejores prácticas* y patrones de diseño para la web
- Preparado para aplicaciones empresariales y adaptable a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo
- Código fácil de leer que incluye comentarios de phpDocumentor y que permite un mantenimiento muy sencillo
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros.
- La capa de internacionalización que incluye Symfony permite la traducción de los datos y de la interfaz, así como la adaptación local de los contenidos.
- La capa de presentación utiliza plantillas y *layouts* que pueden ser creados por diseñadores HTML sin ningún tipo de conocimiento del framework. Los *helpers* incluidos permiten minimizar el código utilizado en la presentación, ya que encapsulan grandes bloques de código en llamadas simples a funciones.
- Los formularios incluyen validación automatizada y relleno automático de datos (*"repopulation"*), lo que asegura la obtención de datos correctos y mejora la experiencia de usuario.
- Los datos incluyen mecanismos de escape que permiten una mejor protección contra los ataques producidos por datos corruptos.
- La gestión de la caché reduce el ancho de banda utilizado y la carga del servidor.
- La autenticación y la gestión de credenciales simplifican la creación de secciones restringidas y la gestión de la seguridad de usuario.
- El sistema de enrutamiento y las URL *limpias* permiten considerar a las direcciones de las páginas como parte de la interfaz, además de estar optimizadas para los buscadores.
- Los listados son más fáciles de utilizar debido a la paginación automatizada, el filtrado y la ordenación de datos.
- Los plugins, las factorías (patrón de diseño *"Factory"*) y los *"mixin"* permiten realizar extensiones a medida de Symfony.
- Las interacciones con Ajax son muy fáciles de implementar mediante los *helpers* que permiten encapsular los efectos Javascript compatibles con todos los navegadores en una única línea de código.

Symfony esta desarrollado tomando lo mejor de otros framework. Basado en el patrón arquitectónico MVC y las mejores practicas de diseño web.

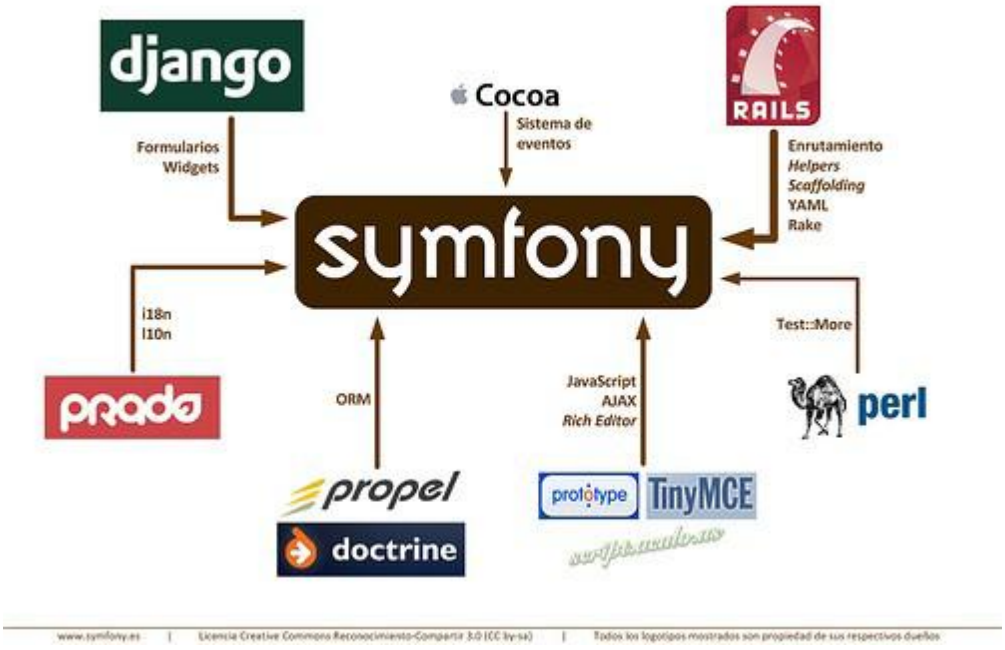


Ilustración 08. Características de Symfony

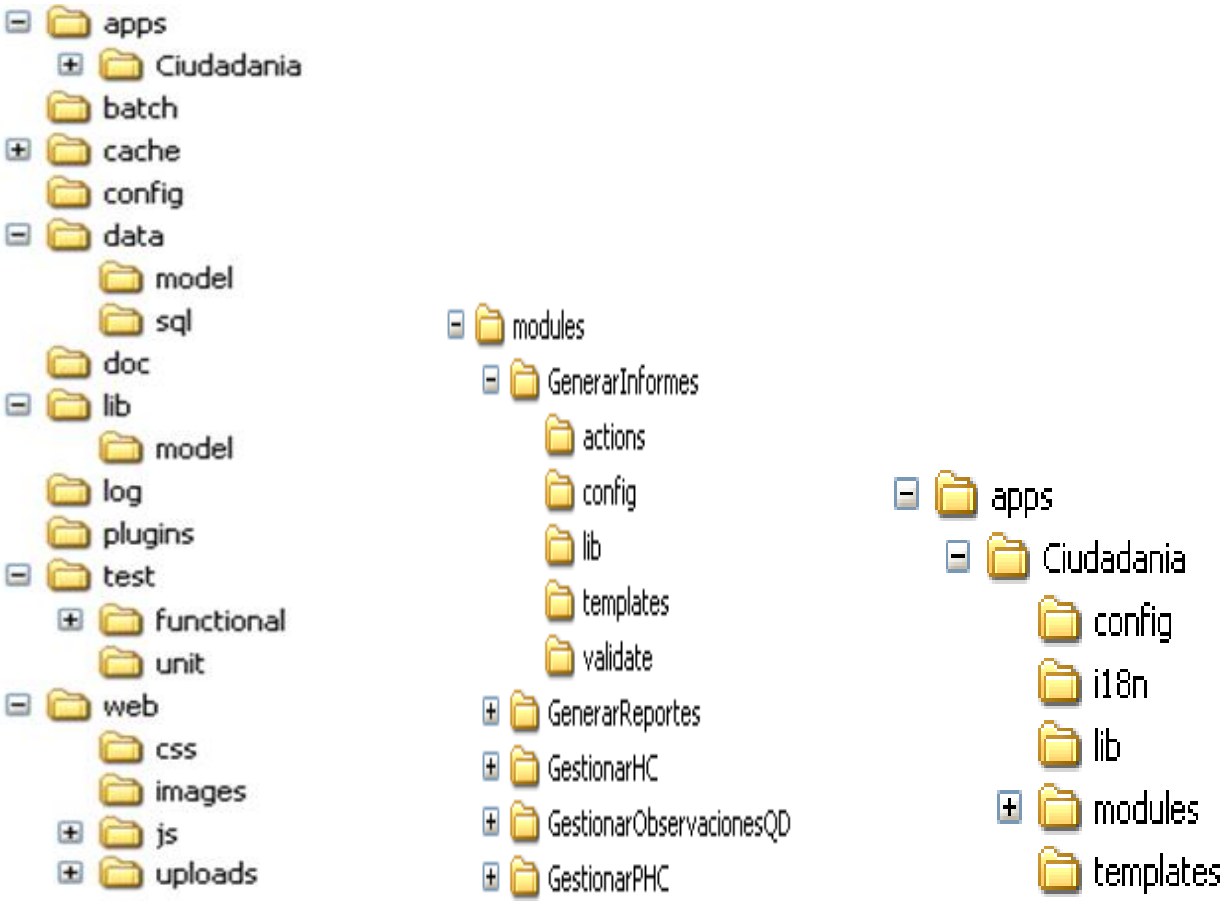
¿Por que usar Symfony?

- Escalable: Symfony es infinitamente escalable si se disponen de los recursos necesarios. Yahoo utiliza Symfony para programar aplicaciones con 20 millones de usuarios y 12 idiomas.
- Probado: Symfony ha sido probado con éxito durante varios años en aplicaciones muy diferentes. Desde sitios web con millones de usuarios (del.icio.us, Yahoo Bookmarks, Yahoo Answers) hasta otros miles de sitios pequeños y medianos.
- Soporte: Symfony sigue una política de tipo LTS (soporte a largo plazo). Las versiones estables se mantienen durante 3 años sin cambios pero con una continua corrección de los errores conocidos. Tus clientes estarán siempre contentos y a ti no te costará nada hacerlo.
- Licencia: Symfony utiliza una licencia MIT, con la que se puede hacer aplicaciones web comerciales, gratuitas y/o de software libre.
- Compromiso: la empresa que ha creado Symfony no vive del framework, sino de las aplicaciones que hace con él. Esto significa que a ellos les interesa tanto como a los usuarios aspectos como el rendimiento, la buena documentación, el soporte muy largo, etc.

- Código: Desde su primera versión Symfony ha sido creado para PHP 5, desechando la versión PHP 4 (que ha sido declarada obsoleta recientemente).
- Seguro: Se puede controlar hasta el último acceso a la información e incluye por defecto protección contra ataques XSS y CSRF.
- Documentado: se trata del framework PHP mejor documentado: miles de páginas en el wiki oficial, tutoriales de hasta 250 páginas y un libro gratuito de casi 500 páginas. Además, el libro está completamente traducido al español.
- Calidad: su código fuente incluye más de 8.000 pruebas unitarias y funcionales.

Modelo del framework (estructura física)

La raíz de cualquier proyecto Symfony contiene los siguientes directorios:



Las siguientes tablas describen cada uno de los subdirectorios.

Tabla 2. Directorios en la raíz de los proyectos Symfony

Directorio	Descripción

apps/	Contiene un directorio por cada aplicación del proyecto.
batch/	Contiene los scripts de PHP que se ejecutan mediante la línea de comandos o mediante la programación de tareas para realizar procesos en lotes
cache/	Contiene la versión cacheada de la configuración y la versión cacheada de las acciones y plantillas del proyecto. El mecanismo de cache utiliza los archivos de este directorio para acelerar la respuesta a las peticiones web.
config/	Almacena la configuración general del proyecto
data/	En este directorio se almacenan los archivos relacionados con los datos, (el esquema de una base de datos, el archivo que contiene las instrucciones SQL para crear las tablas)
doc/	Contiene la documentación del proyecto, formada por tus propios documentos y por la documentación generada por PHPdoc.
lib/	Almacena las clases y librerías externas. Se suele guardar todo el código común a todas las aplicaciones del proyecto. El subdirectorio model/ guarda el modelo de objetos del proyecto.
log/	Guarda todos los archivos de log generados por Symfony.
plugins/	Almacena los plugins instalados en la aplicación
test/	Contiene las pruebas unitarias y funcionales escritas en PHP y compatibles con el framework de pruebas de Symfony. Cuando se crea un proyecto, Symfony crea algunos pruebas básicas
web/	La raíz del servidor web. Los únicos archivos accesibles desde Internet son los que se encuentran en este directorio

Tabla 3. Subdirectorios de cada aplicación Symfony

Directorio	Descripción
config/	Contiene un montón de archivos de configuración creados con YAML. Aquí se almacena la mayor parte de la configuración de la aplicación, salvo los parámetros propios del framework. También es posible redefinir en este directorio los parámetros por defecto si es necesario.
i18n/	Contiene todos los archivos utilizados para la internacionalización de la aplicación, sobre todo los archivos que traducen la interfaz. La internacionalización también se puede realizar con una base de datos, en cuyo caso este directorio no se utilizaría
lib/	Contiene las clases y librerías utilizadas exclusivamente por la aplicación
modules/	Almacena los módulos que definen las características de la aplicación
templates/	Contiene las plantillas globales de la aplicación, es decir, las que utilizan todos los módulos. Por defecto contiene un archivo llamado layout.php, que es el layout principal con el que se muestran las plantillas de los módulos

En las aplicaciones recién creadas, los directorios i18n/, lib/ y modules/ están vacíos. Las clases de una aplicación no pueden acceder a los métodos o atributos de otras aplicaciones del mismo proyecto. Además, los enlaces entre 2 aplicaciones de un mismo proyecto se deben indicar de forma absoluta. Esta última restricción es importante durante la inicialización del proyecto, que es cuando se debe elegir como dividir el proyecto en aplicaciones.

Cada aplicación contiene uno o más módulos. Cada módulo tiene su propio subdirectorio dentro del directorio modules y el nombre del directorio es el que se elige durante la creación del módulo.

Tabla 4. Subdirectorios de cada módulo

Directorio	Descripción
actions/	Normalmente contiene un único archivo llamado actions.class.php y que corresponde a la clase que almacena todas las acciones del módulo. También es posible crear un archivo diferente para cada acción del módulo.
config/	Puede contener archivos de configuración adicionales con parámetros exclusivos del módulo
lib/	Almacena las clases y librerías utilizadas exclusivamente por el módulo
templates/	Contiene las plantillas correspondientes a las acciones del módulo. Cuando se crea un nuevo módulo, automáticamente se crea la plantilla llamada indexSuccess.php
validate/	Contiene archivos de configuración relacionados con la validación de formularios

En los módulos recién creados, los directorios config/, lib/ y validate/ están vacíos.

Existen pocas restricciones sobre la estructura del directorio web, que es el directorio que contiene los archivos que se pueden acceder de forma pública. Si se utilizan algunas convenciones básicas en los nombres de los subdirectorios, se pueden simplificar las plantillas.

Tabla 5. Subdirectorios habituales en la carpeta web

Directorio	Descripción
css/	Contiene los archivos de hojas de estilo creados con CSS (archivos con extensión .css
images/	Contiene las imágenes del sitio con formato .jpg, .png o .gif
js/	Contiene los archivos de Javascript con extensión .js
uploads/	Se almacenan los archivos subidos por los usuarios. Aunque normalmente este directorio contiene imágenes, no se debe confundir con el directorio que almacena las imágenes del

	sitio (images/). Esta distinción permite sincronizar los servidores de desarrollo y de producción sin afectar a las imágenes subidas por los usuarios

Aunque es muy recomendable mantener la estructura definida por defecto, es posible modificarla para adaptarse a las necesidades específicas de cada proyecto, como por ejemplo los proyectos que se ejecutan en servidores con sus propias estructuras de directorios definidas y con otras políticas para el desarrollo de las aplicaciones.

La implementación que realiza Symfony de la arquitectura MVC incluye varias clases. Ejemplo de algunas de las más importantes son:

- sfController: es la clase del controlador. Se encarga de decodificar la petición y transferirla a la acción correspondiente.
- sfRequest almacena todos los elementos que forman la petición (parámetros, cookies, cabeceras, etc.)
- sfResponse contiene las cabeceras de la respuesta y los contenidos. El contenido de este objeto se transforma en la respuesta HTML que se envía al usuario.
- El *singleton* de contexto (que se obtiene mediante sfContext::getInstance()) almacena una referencia a todos los objetos que forman el núcleo de Symfony y puede ser accedido desde cualquier punto de la aplicación.

Todas las clases de Symfony utilizan el prefijo sf, como también hacen todas las variables principales de Symfony en las plantillas. De esta forma, se evitan las *colisiones* en los nombres de clases y variables de Symfony y los nombres de las clases y variables de los desarrolladores, además de que las clases del framework son más fáciles de reconocer.

2.3.2.3 Descripción del marco de trabajo.

El Marco de Trabajo va estar estructurado en tres capas, ya que es el estilo a utilizar dando la posibilidad de dividir en niveles físicos (al utilizar MVC como patron arquitectonico, las capas cooinciden con las de este patrón):

- En la primera capa (vista), se encuentra la capa de presentación donde están los diferentes componentes y estilos, siendo estos Javascript (jQuery), imágenes, estilos css, plantillas (archivos accionSuccess y layout.php) y archivos de validación.

- En la segunda capa (control), capa de lógica de negocio se encuentran las diferentes clases de lógica de negocio (clases encargadas del negocio, ubicadas en las carpetas lib, los archivos action.class.php y los controladores frontales).
- En la capa de acceso a datos (modelo), se van encontrar las configuraciones de las conexiones (archivos de mapeo). De la abstracion de la base de datos se encarga Creole y de la generacion de las clases del modelo se encarga Propel.

La figura muestra la organización del código para el proyecto, siguiendo la estructura proyecto / aplicación / módulo / acción. La estructura de directorios real del proyecto fue mostrada anteriormente.

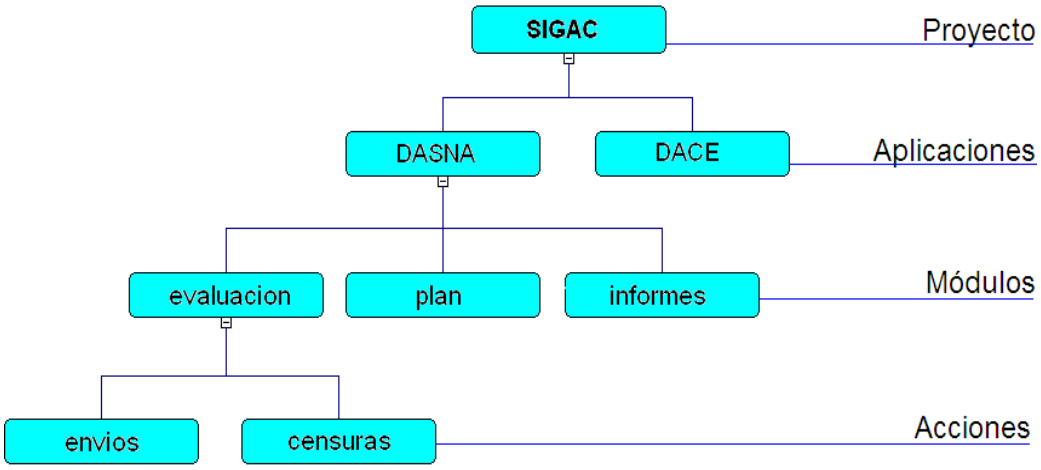


Ilustración 09. Ejemplo de organización del código

En esta otra figura podemos ver la relación entre el modelo físico y el lógico:

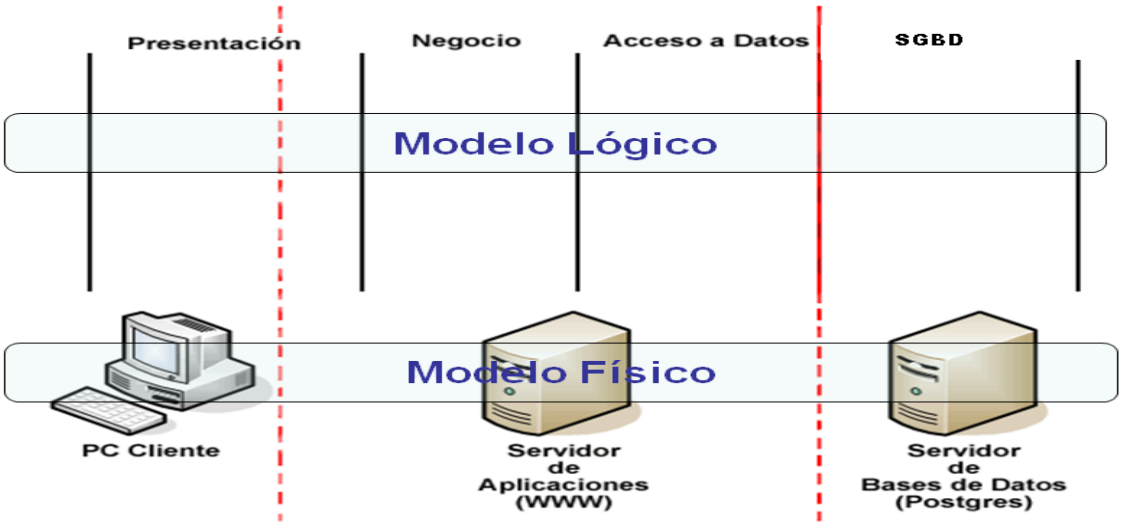


Ilustración 10. Relación entre Modelo lógico y el físico

2.3.3 Patrones.

El establecimiento de los patrones es lo que posibilita el aprovechamiento de la experiencia acumulada en el diseño de aplicaciones. Algunos de los patrones que se ven en SIGAC son (20):

MVC: con este patrón se consigue separación de responsabilidades. Symfony esta desarrollado siguiendo dicho patrón. Anteriormente se explico dicho patrón, así como su uso y relación en la arquitectura del sistema. (21)

ORM: ayuda a cargar y guardar los objetos persistentes que son utilizados por la aplicación. Propel es el ORM utilizado por defecto por Symfony (puede usarse Doctrine) y nos provee de una interfaz genérica acceso a las bases de datos, que nos permite utilizar varios gestores de bases de datos (o cambiar de gestor en cualquier etapa del proyecto, sin que esto afecto el desarrollo del mismo).

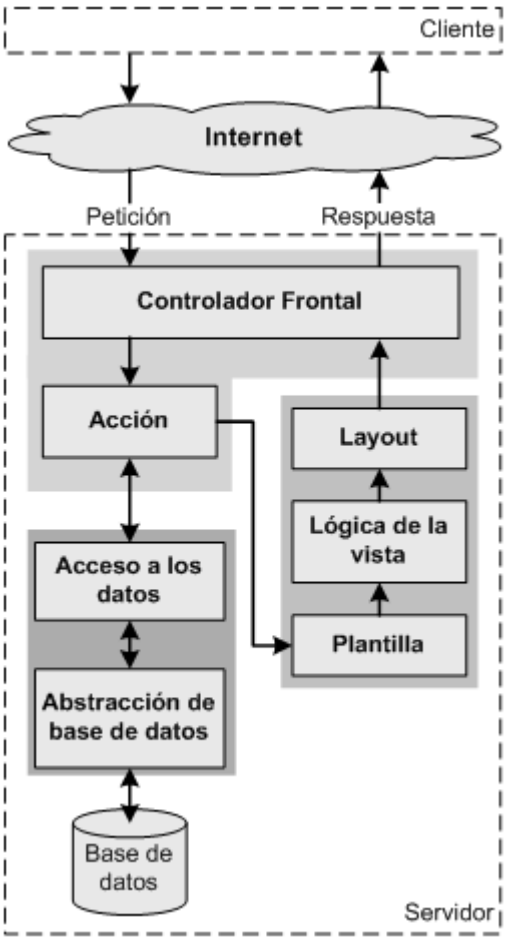


Ilustración 11. El flujo de trabajo del sistema con MVC

Application Controller: este patrón se encarga de gestionar la interacción entre el usuario y la aplicación, dirigiendo el flujo de navegación y controlando el estado de la sesión. Se utiliza para la navegación no lineal, por ejemplo: “si el usuario pincha aquí y es un usuario normal, que vaya a ésta página, pero si pincha un usuario administrador, que vaya a esta otra”. Además de ofrecernos esta característica, nos ofrece control sobre el estado de la sesión del usuario.

Row/Table Data Gateway: consiste en crear una instancia por cada tabla y fila existente en la BD. Es decir cada clase del modelo presenta un mapeo uno a uno a una tabla de la BD y cada objeto o instancia de la clase mapea a una fila o registro de la tabla. Sus métodos consisten en las operaciones básicas que se realizan sobre estas tablas, insertar, modificar y eliminar.

2.3.3.1 Patrones GRASP.

- **Creador:** En la clase Actions se encuentran las acciones definidas para la aplicación y se ejecutan cada una de ellas. En las acciones se crean los objetos de las clases que representan

las entidades, evidenciando de este modo que la clase Actions es "creador" de dichas entidades.

- **Experto:** Uno de los más utilizados, puesto que Propel (librería externa que utiliza Symfony para realizar su capa de abstracción en el modelo), encapsula toda la lógica de los datos y son generadas las clases con todas las funcionalidades comunes de las entidades.
- **Alta Cohesión:** Symfony permite asignar responsabilidades con una alta cohesión, por ejemplo la clase Actions tiene la responsabilidad de definir las acciones para las plantillas y colabora con otras para realizar diferentes operaciones, instanciar objetos y acceder a las propiedades, es decir, está formada por diferentes funcionalidades que se encuentran estrechamente relacionadas proporcionando que el software sea flexible frente a grandes cambios.
- **Controlador:** Todas las peticiones Web son manejadas por un solo controlador frontal (sfActions), que es el punto de entrada único de toda la aplicación en un entorno determinado. Cuando el controlador frontal recibe una petición, utiliza el sistema de enrutamiento para asociar el nombre de una acción y el nombre de un módulo con la URL entrada por el usuario.

2.3.3.2 Patrones GOF.

- **Singleton:** Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. En el controlador frontal hay una llamada a sfContext::getInstance(). En una acción, el método getContext(), un objeto muy útil que guarda una referencia a todos los objetos del núcleo de Symfony.
- **Abstract Factory:** Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando. Cuando el framework necesita por ejemplo crear un nuevo objeto para una petición, busca en la definición de la factoría el nombre de la clase que se debe utilizar para esta tarea.
- **Decorator:** Añade funcionalidad a una clase, dinámicamente. El archivo layont.php, que también se denomina plantilla global, almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el layout, o si se mira desde el otro punto de vista, el layout decora la plantilla.

Ilustración 12. Ejemplo de patrón Decorator

- **Composite:** Permite tratar objetos compuestos como si de uno simple se tratase. Sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol. Esto simplifica el tratamiento de los objetos creados, ya que al poseer todos ellos una interfaz común, se tratan todos de la misma manera.
- **Facade:** que proporciona una interfaz unificada para un conjunto de de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar. Como es el caso del acceso a base de datos donde existe una interfaz común para los diferentes gestores de bases de datos soportados por el ORM.
- **Observer:** proporciona a un objeto un mecanismo para avisar a otros objetos de cambios en el estado de la aplicación. En el caso de una aplicación Web, se puede utilizar para que la aplicación reaccione a los eventos generados por el usuario, ejecutando en reacción a ellos la lógica que se ha definido.

2.4 Tecnología y herramientas de desarrollo.

2.4.1 Sistema operativo.

Para el desarrollo se decidió trabajar sobre Linux:

- Para el cliente: Ubuntu Gutsy Gibbon (7.10). Se utilizara esta distribución por ser un entorno fácil de trabajar, con una amplia gama de software equivalente a los utilizados en Windows, documentación disponible en varios idiomas (entre ellas español), gran comunidad de usuarios, libre, gratis, entre otras.
- Para el servidor: se utilizara Ubuntu Gutsy Gibbon Server Edition (7.10).

Debido a esto las aplicaciones a usar en el desarrollo del proyecto deben ser herramientas libres o con distribuciones para Linux (solo en caso excepcionales se trabajara en Windows con herramientas cuyo similar en Linux, no tenga todas las prestaciones necesarias).

2.4.2 Gestión de proyectos.

Trac: herramienta open source, basado en la web, multiusuario, multiplataforma, permiten la gestión de proyectos, tareas y usuarios, integración con LDAP (autenticación por usuarios del dominio), permiten la creación de diagramas de Gantt (mediante un plugin) y la gestión de documentos. Proporciona, además, integración con el servidor de código subversión, wiki integrada y varias facilidades de

reportes, calendario (timeline) que muestra todos los eventos del proyecto, lo que permite ver el progreso del mismo de forma muy fácil.

2.4.3 Gestión de configuración.

Subversion: repositorio con un único número de versión (identifica todos los archivos del repositorio en cierto punto de tiempo), sigue historial de archivos y directorios a través de copias y renombrados, modificaciones (incluyendo cambios a varios archivos) atómicas, creación de ramas y etiquetas con costo de complejidad constante O (1), envía sólo las diferencias en ambas direcciones, servido mediante Apache sobre WebDAV (permite que clientes WebDAV utilicen subversion en forma transparente), integrado a Apache permite utilizar todas las opciones que este servidor provee en materia de autentificación (SQL, LDAP, PAM, etc.), manejo eficientemente de archivos binarios, permite selectivamente el bloqueo de archivos (se usa en archivos binarios que, al no poder fusionarse fácilmente, conviene que no sean editados por más de una persona a la vez), herramientas graficas (interfaces o clientes) que facilitan el trabajo con él (TortoiseSVN, Subclipse, RapidSVN, KDESvn, SvnWorkbench).

2.4.4 Control, seguimiento y gestión de errores

Trac: optimizado para subversion. Provee una plataforma completa para colaboración entre usuarios y desarrolladores, interfaz de usuario amigable e intuitiva, errores registrados como tareas, crea una entrada en la wiki a cada cambio en el repositorio; seguimiento y documentación de errores, tareas y cambios; integrado con eclipse mediante subclipse y Mylyn, permite asignar errores (archivos) a la tarea correspondiente, además de notificar al usuario de dichas tareas.

2.4.5 Lenguaje de programación.

El lenguaje propuesto para el desarrollo es PHP:

- Es open source.
- Fácil, rápido y ameno de aprender.
- Multiplataforma.
- Documentación amplia, así como disponibilidad de código fuente de casi todo lo que quieras hacer en la web (phpclasees.com es un buen ejemplo).
- Delega todo lo ajeno al procesamiento de código al servidor web.
- Integración con la mayoría de los gestores de bases de datos existentes.
- Gran cantidad de frameworks que facilitan el trabajo del desarrollador web.

- Maneja cada petición como algo individual y llama al PHP solo cuando lo necesita.
- Infinidad de librerías para el trabajo con imágenes, correo, redes, xml, servicios web, matemática, etc.
- Es capas de llamar y ejecutar funciones de otros lenguajes (ejemplo java).
- Permite utilizar código estructurado donde la OOP no trabaja bien.
- Se integra con varios servidores web, aunque la mejor combinación es con Apache.
- Junto con Apache, MySQL y Linux forma la famosa LAMP, uno de los principales responsables del desarrollo de la web 2.0

También se propone la utilización de XML (fundamentalmente para transferencia de datos, servicios web).

2.4.6 Entorno(s) de Desarrollo Integrado.

Entre los que propone la comunidad de software libre esta el eclipse con el plugin PDT (eclipsePDT), desarrollado por un equipo de del proyecto eclipse y de la Zend. Presenta un entorno amigable, resaltado de sintaxis, completamiento de código, alta integración con el framework Symfony, gran cantidad de plugin que le extienden la funcionalidades tales como Aptana (para el desarrollo con AJAX), subclipse para integración con subversion, Mylyn para integración con trac, que hace al eclipse una herramienta de desarrollo potente.

2.4.7 Gestor de base de datos

El gestor propuesto para el desarrollo del proyecto es PostgreSQL:

- Alta concurrencia mediante acceso concurrente multiversión (MVCC), lo que permite a un proceso escribir y a otros acceder a la misma tabla sin necesidad de bloqueos. Esta estrategia es superior al bloqueo por tablas o por filas común en otros gestores (MySQL por ejemplo), eliminando la necesidad del uso de bloqueos explícitos.
- Soporte nativo para gran variedad de tipos de datos. Los usuarios pueden crear sus propios tipos de datos.
- Permite la herencia entre tablas, lo que facilita el desarrollo orientado a objetos.
    - Varios lenguajes procedurales (C, C++, pl/Perl, plPHP, pl/Phyton, Java PL, pl/sh, pl/Tcl), lo que permite utilizar la potencia de dichos lenguajes (desde bifurcaciones y bucles, hasta OOP o programación funcional) dentro del gestor.
- Es rápido y seguro.
- Soporta vistas, triggers, cursores, claves ajenas, consultas anidadas o subselect.

- Es multiplataforma

2.4.8 Servidor de aplicaciones y/o web

Se utilizará el servidor web **Apache**. Como se puede apreciar en el grafico, el 50% de los sitios de internet lo usan como servidor .

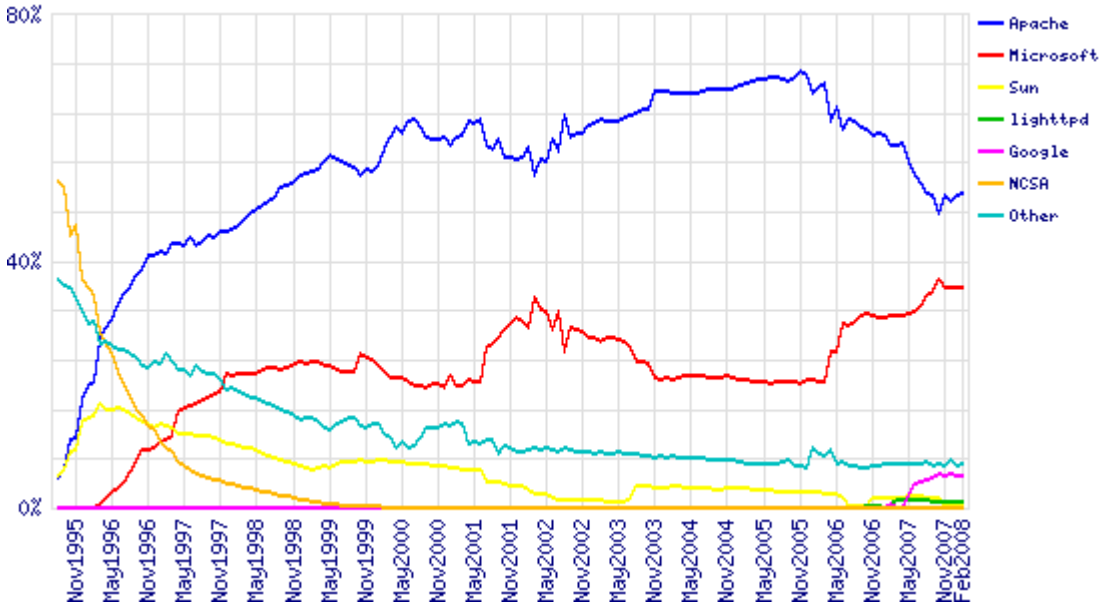


Ilustración 13. Uso de los servidores web más populares. Fuente netcraft.com

Características como los mensajes de error altamente configurables, bases de datos de autenticación y negociado de contenido, modular, open source, multiplataforma, extensible, gratuito y el gran numero de módulos (mod_ssl para comunicaciones seguras vía TLS, mod_rewrite para reescritura de direcciones, mod_dav para soporte del protocolo WebDA, mod_auth_ldap para autentificar usuarios contra un servidor LDAP, mod_php para “correr” PHP como modulo y no como CGI, mod_security para el filtrado a nivel de aplicación, etc.), así como un proceso de instalación y configuración sencillo, lo han convertido en el servidor preferido de muchos.

2.5 Estándares y políticas corporativas adaptadas al sistema.

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

Se encuentran divididos de la siguiente manera:

Estándares de Diseño.

La página principal de la aplicación, se concibe como un portal, que permite la autentificación del sistema. Las páginas deben tener una cabecera (banner) representativa, un área de trabajo, con los formularios correspondientes y una barra menú con las opciones. Se trabaja con una hoja de estilo en común para lograr la uniformidad, donde se utiliza la familia de fuentes Verdana y Arial el tamaño de la misma no debe diferir mucho de 11 a 14 px. Los colores con los que se trabajarán serán tonalidades claras y fuertes basadas en el azul, combinados con el color blanco y gris.

Estándares de Codificación.

Se han realizado una descripción de los estándares de codificación a seguir para implementar el Sistema para el lenguaje PHP y el Gestor PostgreSQL. Para obtener esta información se hace necesario ver el Anexo 1.

Estándares de Organización.

Estos estándares surgen para darle organización a la hora de la elaboración del Sistema. Ya que se tiene definida la estructura de las carpetas en el sitio (estructura propuesta por el framework Symfony), los nombres de los formularios, entre otros.

2.6 Conclusiones.

Se conforma la arquitectura a utilizar en el Sistema con la conjugación de los diferentes estilos arquitectónicos y patrones, los mismos han permitiendo, la organización y diseño a la hora de estructurar la arquitectura. Vimos en este capitulo además las principales características de las herramientas de desarrollo del software, así como la estructura básica del framework de desarrollo.

CAPITULO 3. DESCRIPCIÓN DE LA ARQUITECTURA

3.1 Introducción.

Este capitulo ofrece una visión general de la arquitectura del sistema, usando para esto diferentes vistas arquitectónicas para describir los aspectos más significativos del mismo. Se desea transmitir las decisiones más importantes que afectan o crean la arquitectura del sistema, además que sirva como referencia y le ofrezca un mayor entendimiento del mismo a las partes interesadas.

En este documento se representa la arquitectura del sistema como una serie de vista: vista caso de uso, vista lógica, vista de despliegue, vista de implementación y vista de datos. Estas vistas son representadas con UML, y son modeladas utilizando Visual Paradigm.

3.2 Objetivos y Restricciones Arquitectónicas.

3.2.1 Apariencia o interfaz externa:

- La interfaz a implementar debe ser sencilla para disminuir el tiempo de capacitación de los usuarios finales (principalmente aquellas personas que no son expertas en la rama de la informática).
- Por el uso diario y constante que tendrá el software, la interfaz debe ser agradable, que favorezca el estado de ánimo del cliente y que combine correctamente los colores, tipo de letra y tamaño y que los iconos estén en correspondencia con lo que representan.

3.2.2 Usabilidad:

- El sistema debe ser de fácil manejo para los usuarios que tengan niveles básicos sobre la computación o hallan trabajado con la Web y contar con una ayuda con instrucciones de tipo paso a paso, para entender el trabajo del sistema, así como un listado de definiciones para términos y acrónimos del mismo.

3.2.3 Rendimiento:

- La aplicación debe estar concebida para el consumo mínimo de recursos.
- Un total de 400 usuarios conectados de forma simultanea al servidor central en cualquier momento de tiempo dado.
- Debe completar las transacciones en un tiempo de 60 segundos.
- La latencia del sistema no debe ser mayor de 15 segundos.

- Los clientes no necesitaran más de 128MB de RAM, lo suficiente para ejecutar un navegador web.

3.2.4 Soporte:

Para el servidor de aplicaciones:

- Se requiere que esté instalado PHP en su versión 5.2.5 o superior con las extensiones curl, php_gd2, php_ldap, php_mcrypt, php_openssl, php_pdo, php_pdo_pgsql, php_pgsql, php_soap, php_sqlite, xsl habilitadas.
- Servidor web apache en su versión 2.0.50 o superior con los módulos ssl, rewrite habilitados.

Para el servidor de base de datos:

- Se requiere que esté instalado el gestor de base de datos PostgreSQL 8.2.1 o superior.

Para el cliente:

- Se requiere esté instalado uno de los siguientes navegadores web: Internet Explorer 5.5 o superior, Mozilla Firefox 2.0.0.1 o superior, Opera 9 o superior, Safari 2 o superior.
- Se requiere que los navegadores tengan habilitado el Javascript.

3.2.5 Portabilidad, Escalabilidad, Reusabilidad:

- El sistema será multiplataforma.
- La aplicación se construirá utilizando patrones (de diseño como los GRAPS y GOF) y estándares internacionales de implementación, documentación y diseño (XML, SOAP, WSDL, normas ISO), para facilitar su integración futura, con componentes desarrollados por cualquiera de las partes y garantizar posibilidades de mantenimiento ágil y seguro.
- El sistema deberá poder ser instalado en cualquier Sistema Operativo.
- Debido a los cambios en las condiciones económicas del país, las empresas cubanas toman decisiones continuas que cambian las condiciones en que se desarrollan los procesos, por lo que el sistema deberá implementar la forma de adaptarse ante el cambio de dichas condiciones.
- Deberá implementar servicios (RR.HH) que estén a la escucha del pedido de información de otros sistemas que en el futuro requerirán información.

3.2.6 Hardware:

Para las estaciones de trabajo:

- Se requiere tengan tarjeta de red.
- Se requiere tengan al menos 128 MB de memoria RAM.
- Se requiere al menos 100MB de disco duro.
- Procesador 800 MHz como mínimo.

Para los servidores:

- Se requiere tarjeta de red.
- Se requiere tenga la menos 512MB de RAM.
- Se requiere al menos 40GB de disco duro.
- Procesador 2.0 GHz como mínimo.

Dispositivo: Impresora

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

3.2.7 Software:

- En las computadoras de los clientes se garantizará versiones de Windows 2000 o superior, así como Linux y sus correspondientes distribuciones.
- En las computadoras de los clientes solo se requiere de un navegador (Internet Explorer versión 4.5 o superior, Mozilla Firefox versión 2.0.0.1 o superior, Opera 9 o superior, Safari 2.0 o superior).

3.2.8 Seguridad:

- El sistema debe poder comunicarse usando un protocolo seguro (https).
- Los datos que no pueden viajar de forma transparente por la red, deben ser encriptados.
- Chequear si el usuario que está accediendo al sistema está autenticado y brindarle servicio de autenticación.
- Permitir que cuando se borre cualquier documento o información pueda existir una opción de advertencia antes realizar la acción.
- Realizar auditoria a los principales eventos dentro del sistema, registrando al usuario, el tipo de usuario y los eventos efectuados.

- Un porciento de la seguridad corre por parte del lenguaje y Framework propuesto (PHP y Symfony respectivamente) y otra parte por parte de los servidores (Apache, LDAP, PostgreSQL).
- Se implementará un mecanismo de acceso a la base de datos, que está dado por la diferenciación de las acciones que el sistema realiza en cada momento. Es decir, un usuario para lectura-escritura cuando se requiera modificar y acceder a los datos y otro con los privilegios administrativos, para la realización de copias de seguridad y otras acciones administrativas.
- La asignación de roles a los usuarios y sus funcionalidades sobre el sistema se definirán desde el modulo de Administración.
- Se utilizará reglas o principios de la "programación segura" (diseño simple y abierto, separación de privilegios, control de acceso apropiado, validación de datos de entrada y salida, tratamiento de errores, utilización de criptografía, reutilización de código, control del flujo de datos, control de sobrecarga del búfer, control de inyección de código).
- Programación de disparadores, funciones o procedimientos almacenos y reglas en la Base de Datos para no permitir la manipulación de los datos directamente en el SGBD.
- Debe quedar constancia de quién, desde donde, y cuando se realizó una operación determinada en el sistema.

3.2.9 Confiabilidad, Integridad, Fiabilidad:

- La información manejada por el sistema está protegida de acceso no autorizado y divulgación.
- La información manejada por el sistema será objeto de cuidadosa protección contra la corrupción de los datos y accesos indebidos.
- Debe garantizarse el resguardo de la información (imágenes, documentos), así como la grabación periódica (backups) de la Base de Datos, de forma tal que se posibilite la reinstalación del sistema y los datos, en caso de fallos en el sistema o en el hardware.

3.2.10 Legales:

- El sistema se basa en el manual de normas y principios establecidos por el MAC.
- La mayoría de las herramientas de desarrollo son libres y del resto, las licencias están avaladas.
- El sistema tendrá en cuenta lo establecido por el 'Reglamento de las funciones de las Direcciones y Delegaciones del Ministerio de Auditoria y Control'', en todo lo referido al desarrollo del sistema.

3.2.11 Redes:

- La red existente en las instalaciones debe de soportar la transacción de paquetes de información de al menos unas 36 máquinas a la vez en las delegaciones y 350-400 en el organismo central.
- La transmisión se implementará utilizando el protocolo TCP/IP entre los servidores (FTP, Bases de Datos y web) y mediante el protocolo HTTP/S entre los clientes y el servidor, lo que garantizaría en ambos casos la trasmisión correcta de los datos.
- Para hacer más fiable la aplicación debe de estar protegida contra fallos de corriente y de conectividad, para lo que se deberá parametizar los tiempos para realizar copias de seguridad.

3.3 Vista de Casos de Uso.

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

Modulo Dirección de Planificación Análisis y Control (DPAC): se encarga de la Gestión del plan anual de trabajo del ministerio, la emulación, categorización, así como de la gestión del registro nacional de auditores.

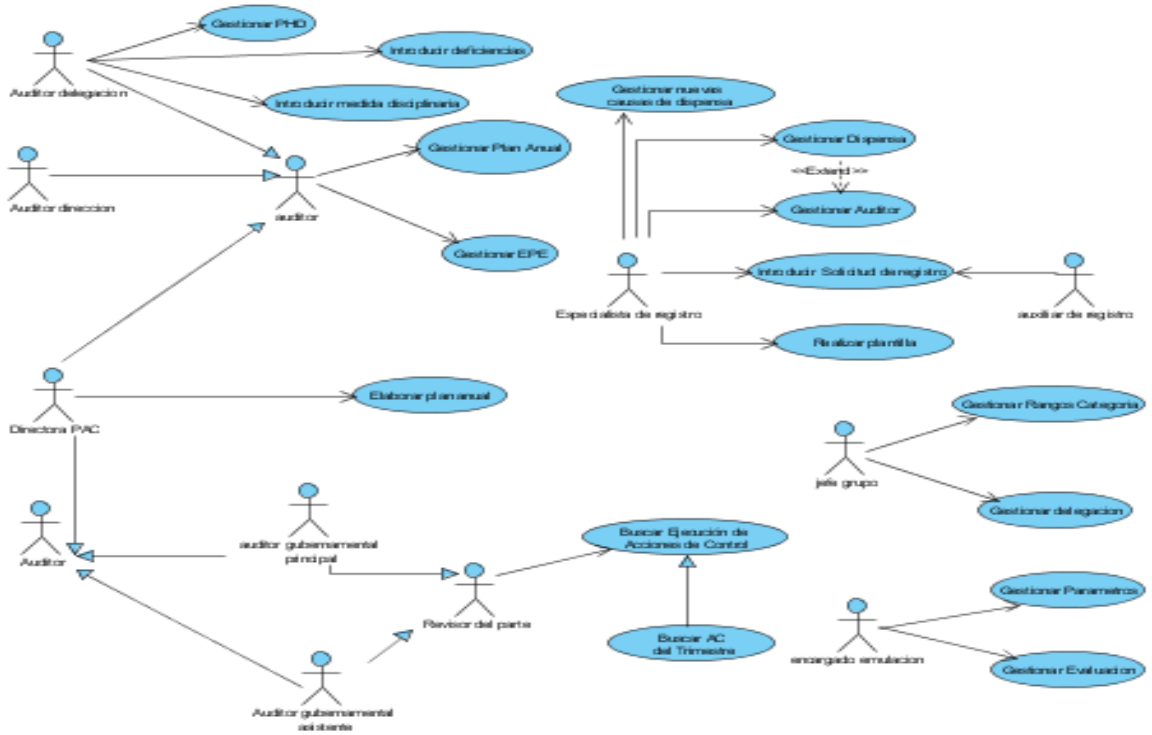


Ilustración 14. Casos de uso Modulo DPAC

Modulo Direccion de Atencion a la Ciudadania (DAC): se encarga de la gestion de las quejas o denuncias de la reportadas por la poblacion asi como de la gestion de los presuntos hechos de corrupcion.

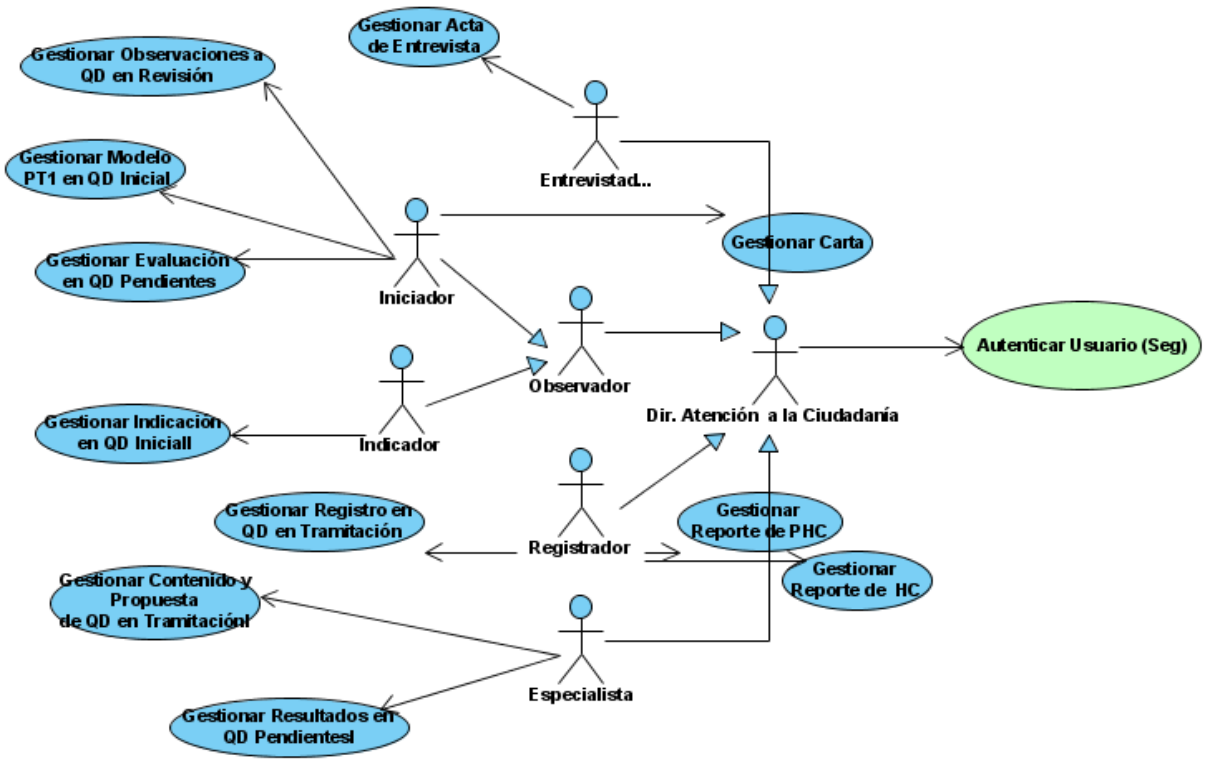


Ilustración 15. Casos de uso Modulo DAC

Modulo Direccion de Auditorias y Controles Especiales (DACE): se encarga de la gestion y ejecucion de las auditorias y controles especiales, asi como de los procesos de fiscalizacion.

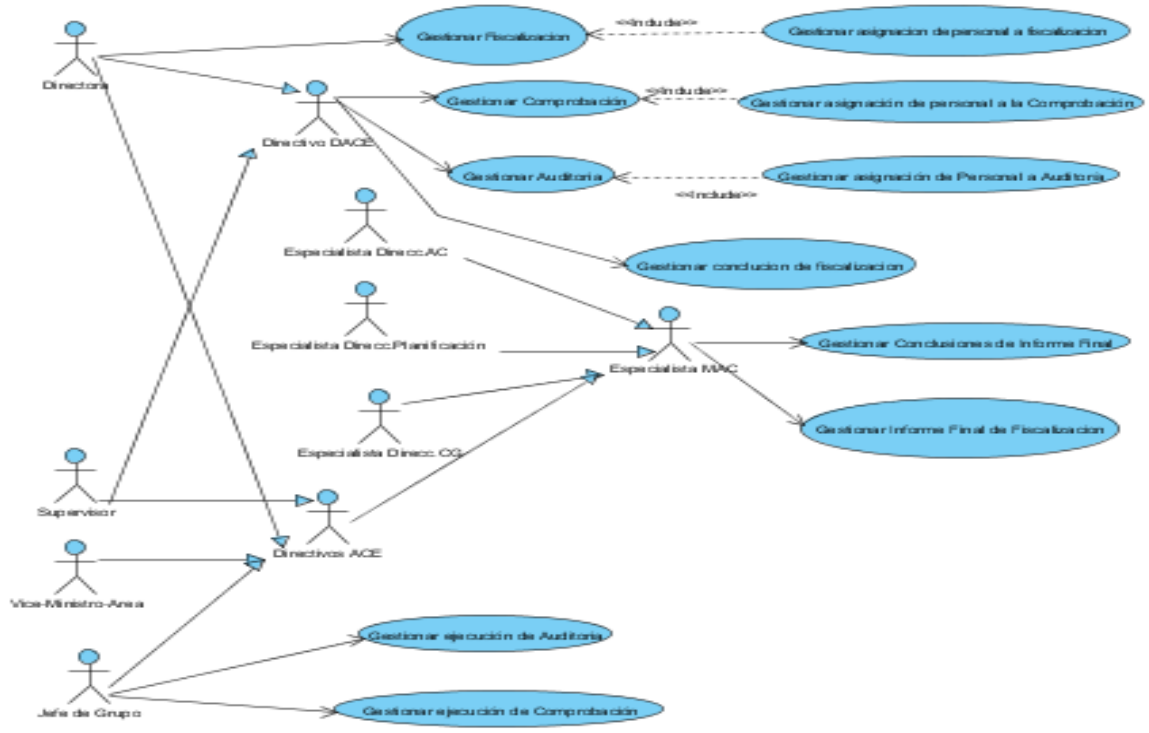


Ilustración 16. Casos de uso Modulo DACE

Modulo Direccion de Auditorias Gubernamentales (DAG): se encarga de la gestion de la auditorias gubernamentales, asi como la evaluacion de los auditores.

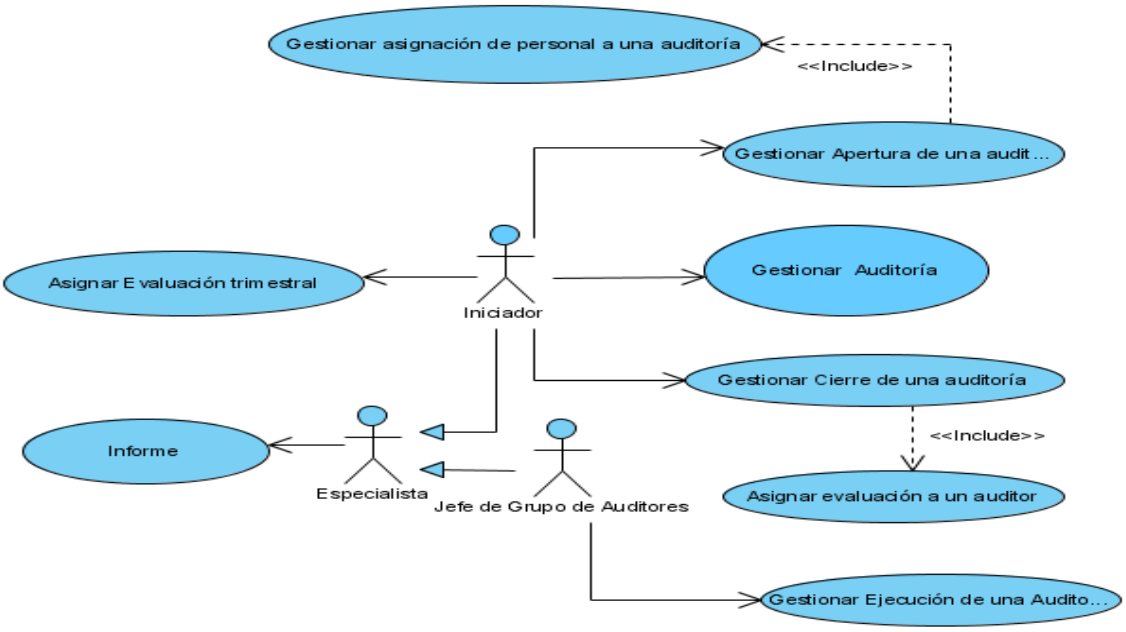


Ilustración 17. Casos de uso Modulo DAG

Modulo Direccion de Atencion al Sistema Nacional de Auditoria (DASNA): encargada de la evaluacion de la calidad de las auditoria, asi como la formacion emergente de auditores.

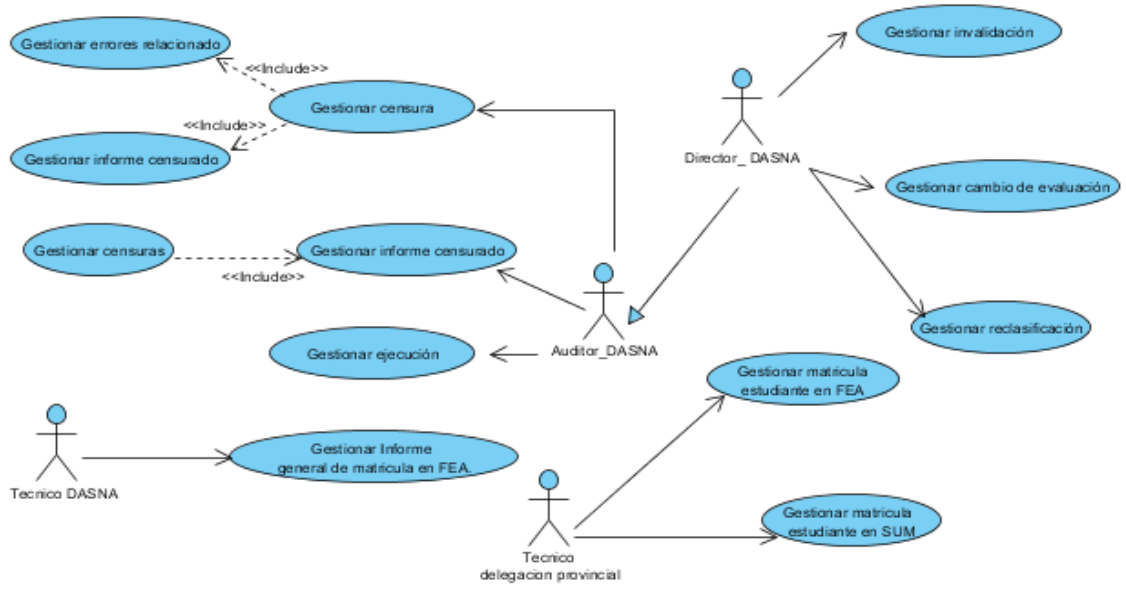


Ilustración 18. Casos de uso Modulo DASNA

Modulo Direccion de Control Gubernamental (DCG): encargada de gestionar y ejecutar los controles gubernamentales a empresas que realizan actividades de comercio exterior, asi como el control de las mismas (directorio).

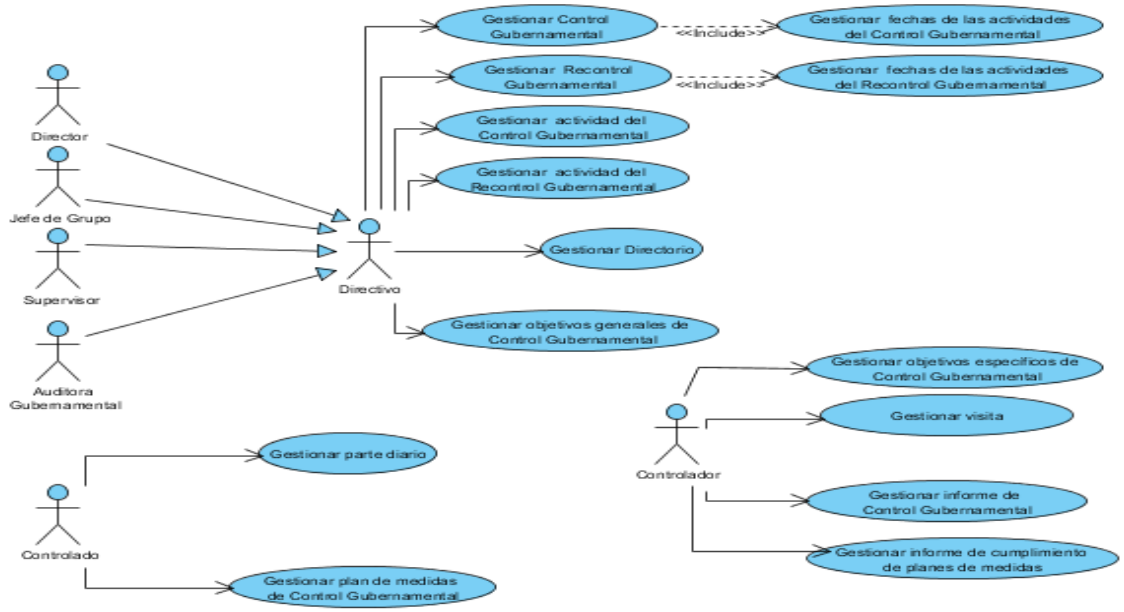


Ilustración 19. Casos de uso Modulo DCG.

3.4 Vista Lógica.

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

La siguiente figura ilustra la división en módulos del sistema a construir, esto se hace con el objetivo de hacer el sistema más reusable y facilitar el mantenimiento ya que cualquier cambio en los procesos del negocio se tendría que cambiar solo en el módulo al que pertenece la funcionalidad que debe cambiar, facilita también el trabajo del equipo de desarrollo ya que permite que se puedan ir desarrollando módulos paralelamente y saber cual es la dependencia entre los mismos.

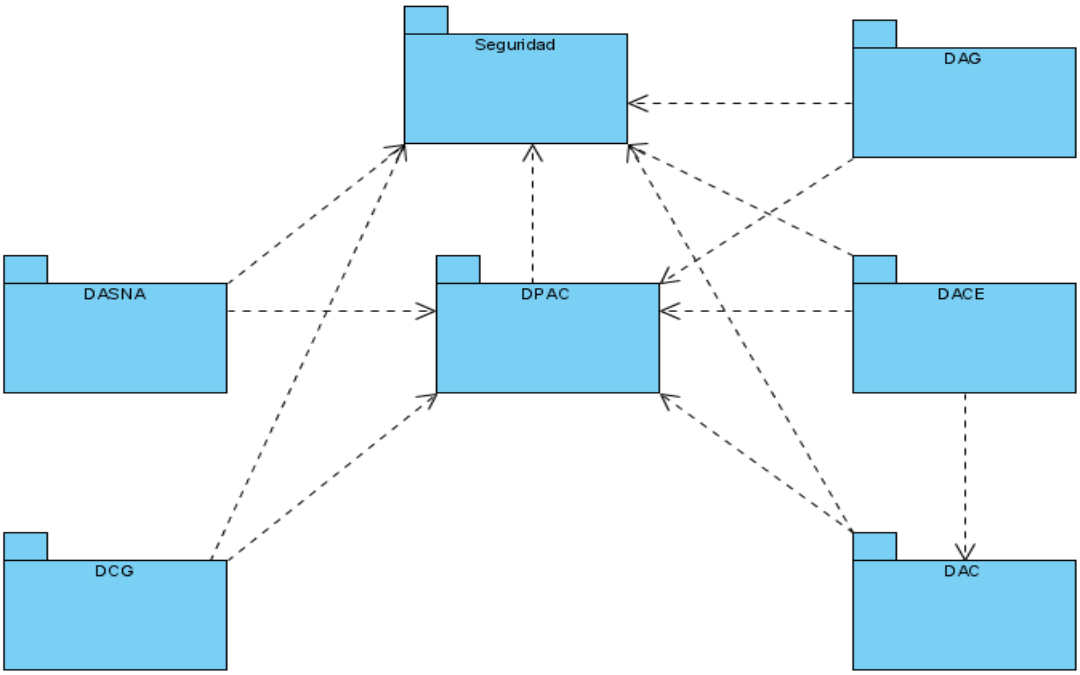


Ilustración 20. Módulos a desarrollar. Dependencias

Paquetes del diseño arquitectónicamente significativos.

A continuación se muestran los paquetes arquitectónicamente significativos durante el diseño, una pequeña descripción de lo que contiene cada paquete y la representación del diagrama de clases del diseño con un ejemplo de uno de los casos de uso del sistema (gestionar conclusiones del paquete DACE), separados por capas.

Aclaración: en lo adelante se trabajara con este caso de uso para ejemplificar como quedarían los diferentes modelos ya que realizar la representación de todos nos llevaría mucho tiempo y espacio (son algo mas de 50 casos de uso), lo que atentaría con el objetivo de este trabajo.

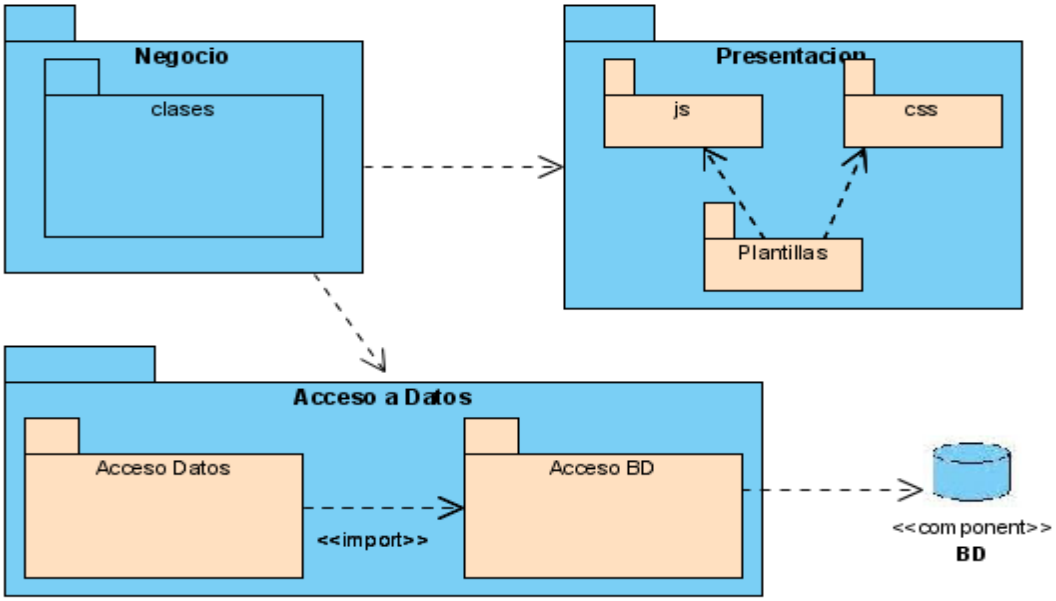


Ilustración 21. Paquetes del diseño arquitectónicamente significativos

Presentación: Contiene los paquetes plantillas, js y css. En el paquete plantillas encontramos las paginas clientes, cuya función es mostrar la información (incluyen formularios para introducir datos en caso de que sea necesario). El paquete css contiene las hojas de estilo asociado a cada caso de uso, así como el paquete js contiene los script que se ejecutaran en el cliente en de cada caso de uso.

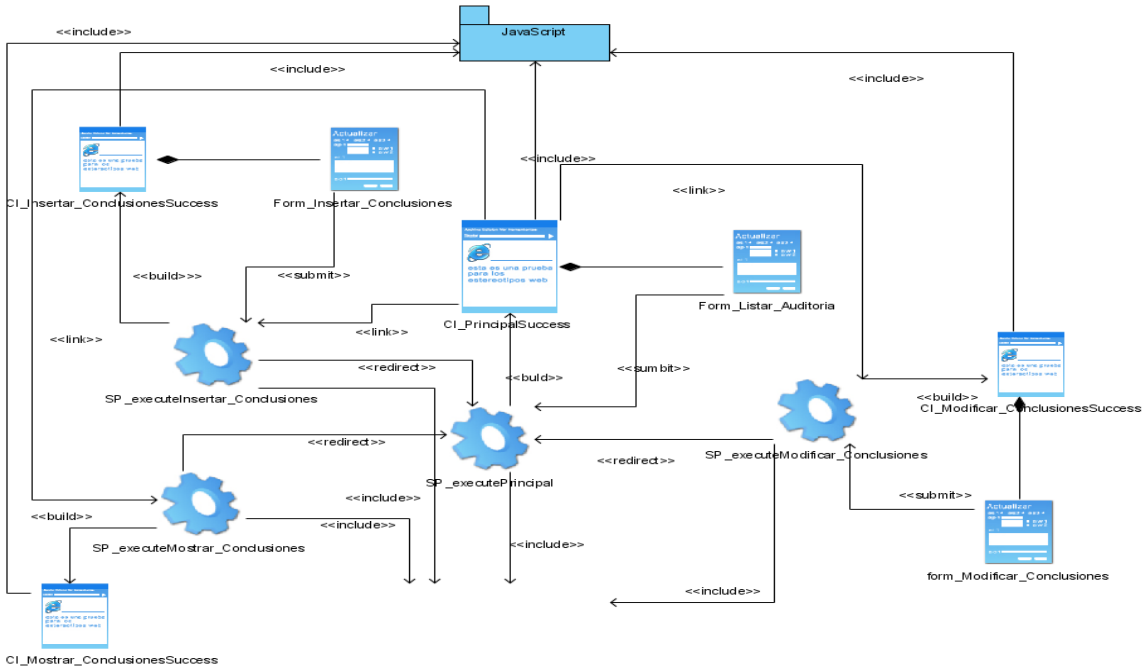


Ilustración 22. Clases del diseño. Presentación

Negocio: Contiene prácticamente es una sola clase (action.class.php) con la que se modela. Aquí se incluyen las clases del Symfony y las clases definidas por los programadores.

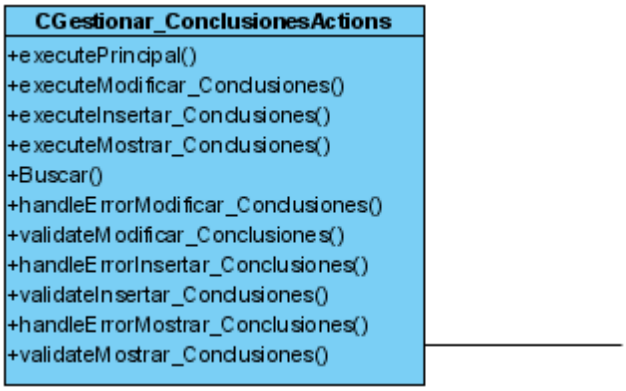


Ilustración 23. Clases del diseño. Negocio

Acceso a Datos: compuesta por dos paquetes Acceso Datos (nos presenta las clases con las que interactuaremos en la aplicación y serán representadas) y Acceso BD (son las clases de acceso a la base de datos directamente, con estas clases no se interactúa).

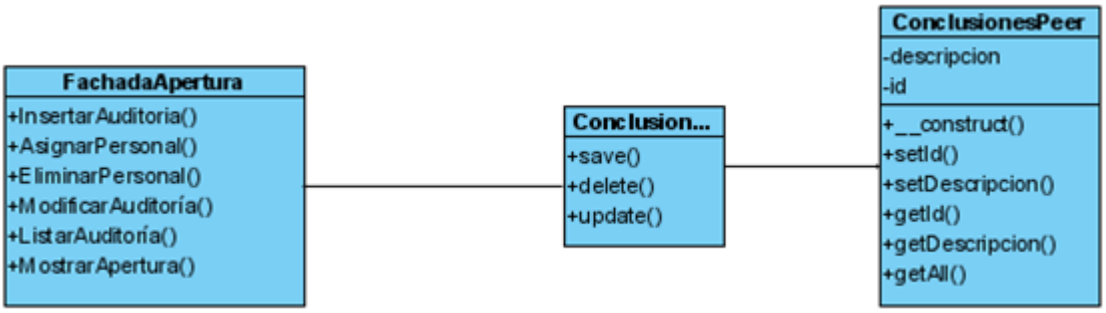


Ilustración 24. Clases del diseño. Acceso a Datos

Realizaciones del caso de Uso.

Se describe como funcionaría el sistema, cómo se llevarían a cabo en términos de clases, dos secciones del mismo.

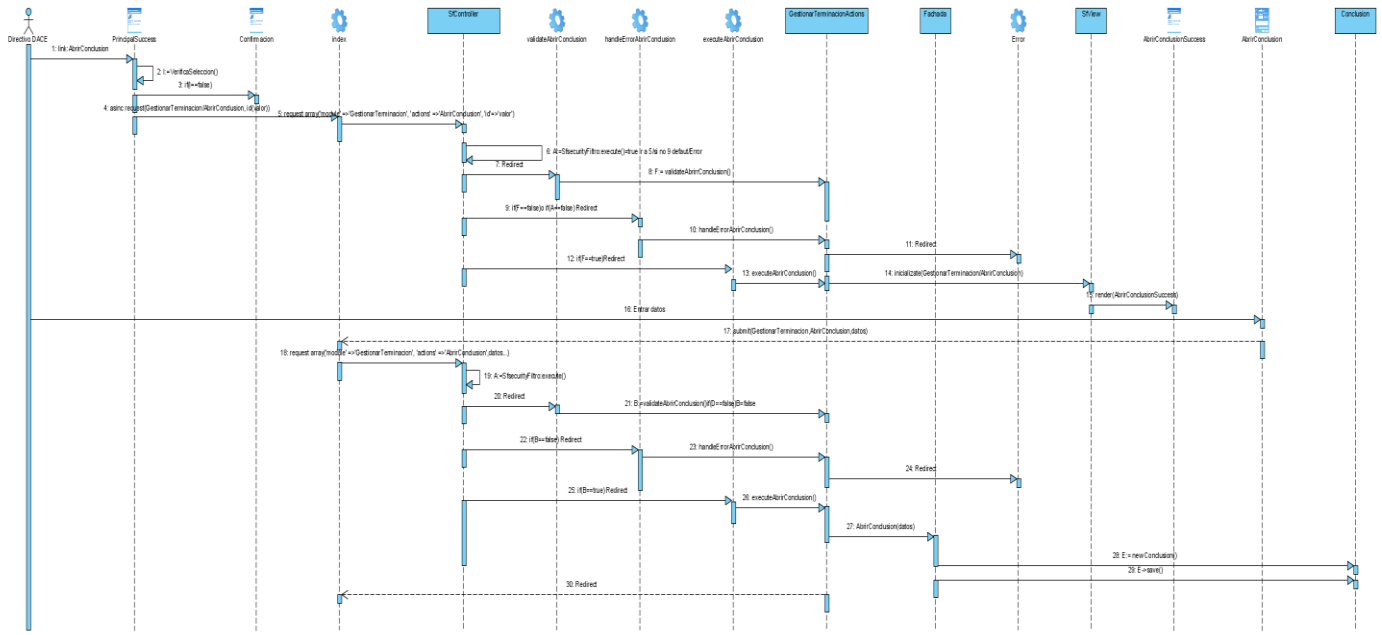


Ilustración 25. Diagrama de secuencia. Insertar

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

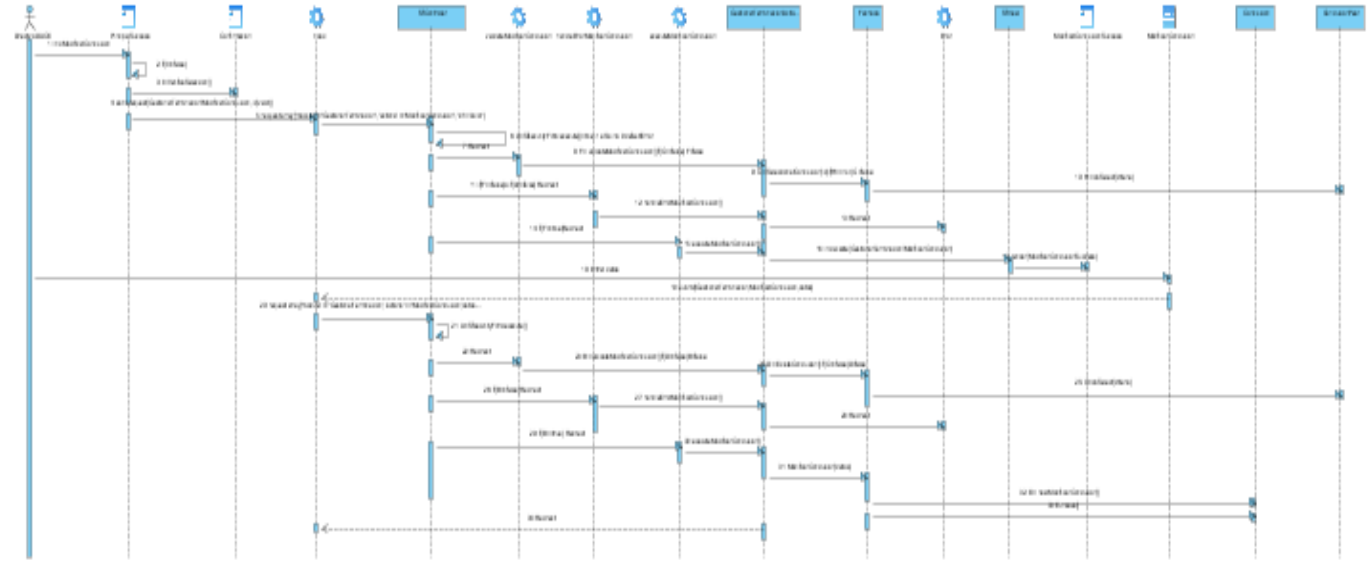


Ilustración 26. Diagrama de secuencia. Editar

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

3.5 Vista de Implementación.

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

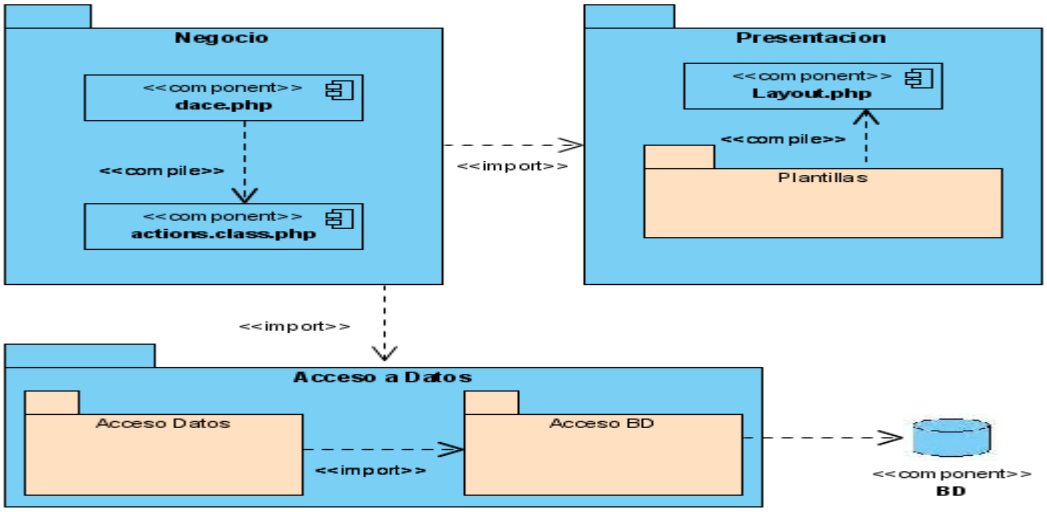


Ilustración 27. Paquetes significativos de la implementación

A continuación se muestra como quedaría el diagrama de componentes para cada una de las capas:

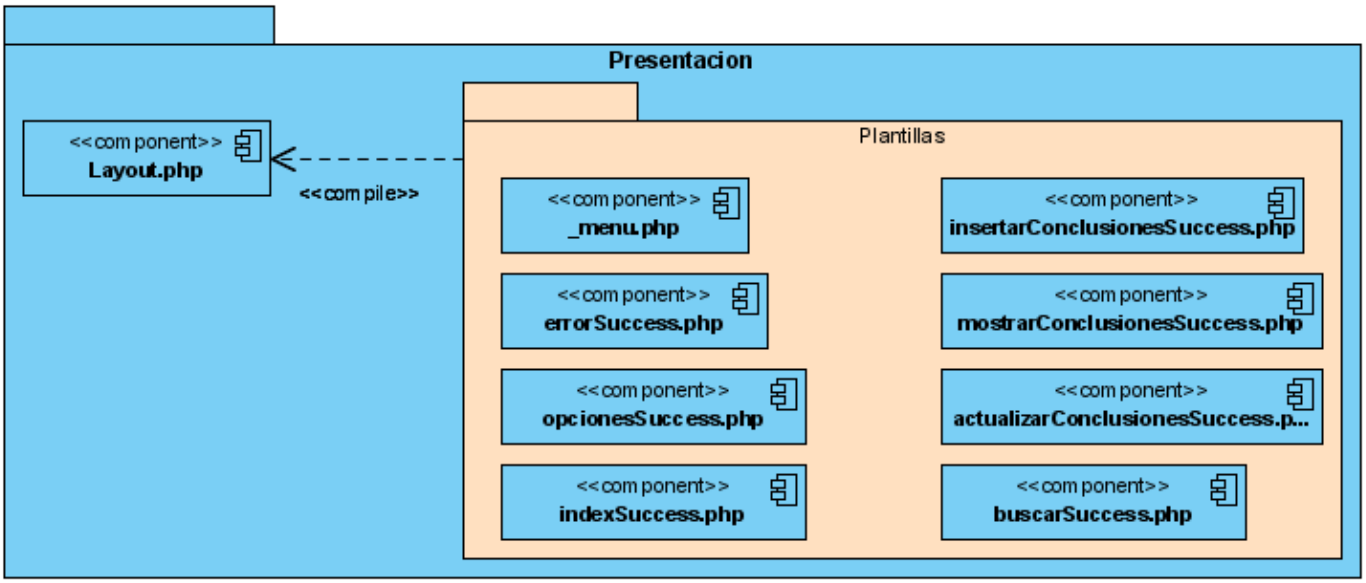


Ilustración 28. Implementación. Presentación

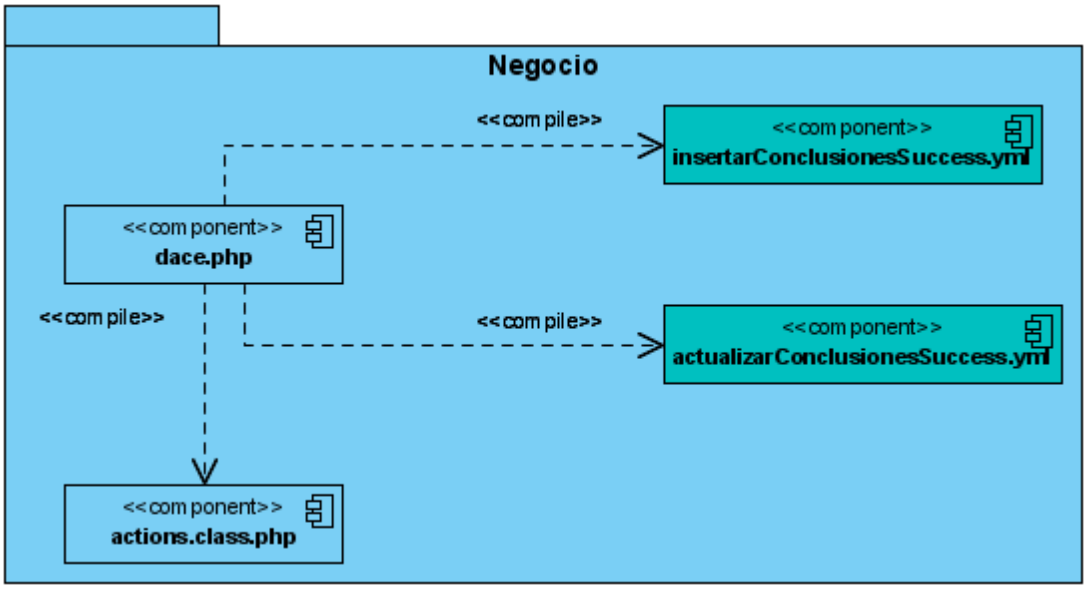


Ilustración 29. Implementación. Negocio

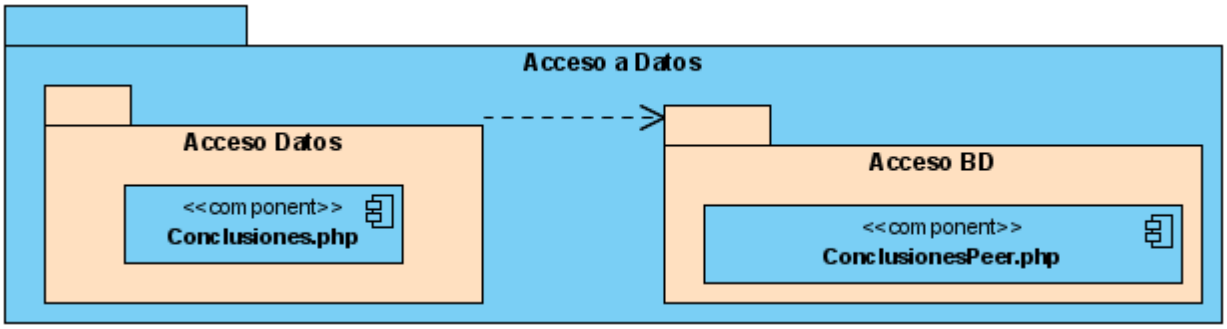


Ilustración 30. Implementación. Acceso a datos

3.6 Vista de Despliegue.

La vista de despliegue propone la distribución física del sistema a través de nodos, es decir modelar la configuración en funcionamiento del sistema (software y hardware) y las relaciones entre sus componentes.

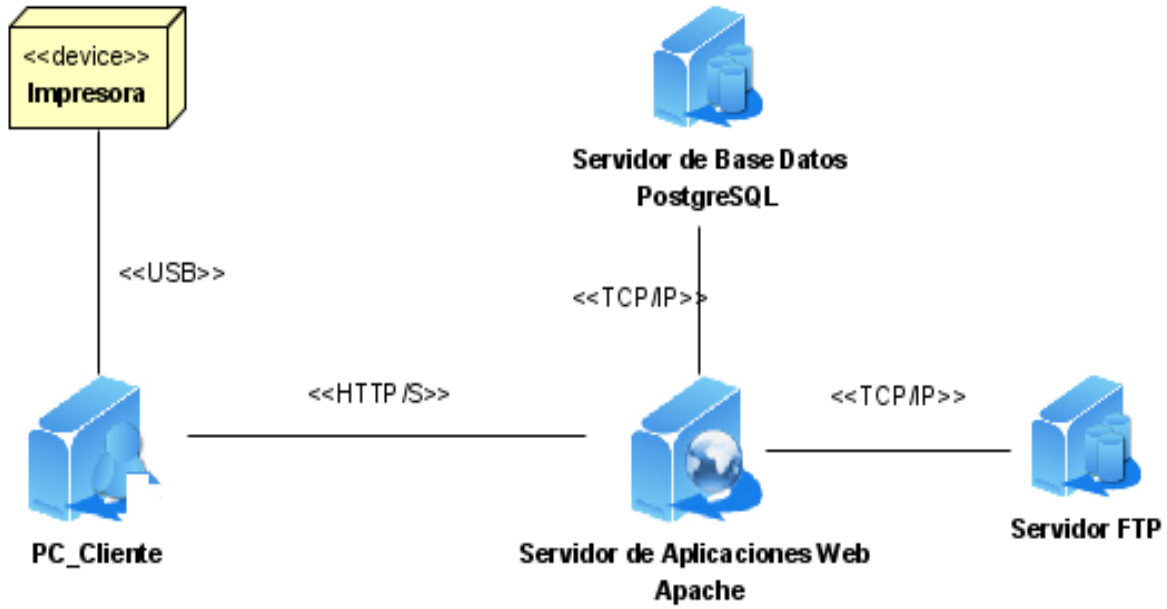


Ilustración 31. Vista de despliegue.

Descripción de los nodos.

Impresora: Satisfacen las necesidades de los clientes de impresión de reportes generados por la aplicación. Las mismas estarán conectadas por USB.

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 Apache: Constituye uno de los nodos fundamentales, en el se encuentra la aplicación y el servidor web apache. Instalado en sistema operativo Linux todos los nodos clientes deben poder acceder al mismo.

Servidor de Base Datos PostgreSQL: Representa el servidor de Bases de Datos. En el se encuentra el gestor PostgreSQL.

Servidor FTP: En este nodo se guardan las imágenes y documentos generados por la aplicación, tales como informes y expedientes de auditorias.

Descripción de elementos e interfaces de comunicación

USB: Representa la comunicación entre la impresora y la PC cliente.

HTTP/S: Protocolo de acceso web. Representa la comunicación entre la PC cliente y el servidor de aplicación.

TCP/IP: Representa la comunicación entre los servidores aplicación-base datos y aplicación-ftp.

3.7 Conclusiones.

En este Capitulo luego de realizar una representación de las diferentes Vistas Arquitectónicas, permitiendo describir la arquitectura, se puede concluir, que las mismas son un subconjunto de modelo de diseño de software, incluyendo los elementos significativos de la arquitectura y excluyendo el diseño de los *componentes* básicos, resuelve a la hora de tomar decisiones sobre qué desarrollar y qué reutilizar, facilitándole un mejor desempeño y entendimiento para el resto de los desarrolladores que tienen como tarea de darle terminación al producto.

CAPITULO 4. EVALUACIÓN ARQUITECTÓNICA

4.1 Introducción.

Los requerimientos no funcionales definen los escenarios, las tácticas a utilizar y los frameworks de desarrollo, todo en función de satisfacer los atributos de:

- Funcionamiento
- Disponibilidad
- Modificabilidad
- Seguridad
- Verificabilidad (testeability)
- Gestionabilidad
- Usabilidad

Hacer un adecuado balance entre la mantenibilidad, la interoperabilidad, la portabilidad, la funcionalidad, la seguridad, la disponibilidad, y la reusabilidad, entre otros atributos, es esencial para obtener un sistema que cumpla las expectativas de las distintas partes interesadas.

4.2 Atributos de calidad.

La calidad de software se define como el grado en el cual el software posee una combinación deseada de atributos. Tales atributos son requerimientos adicionales del sistema, que hacen referencia a características que éste debe satisfacer, diferentes a los requerimientos funcionales. (22) Estas características o atributos se conocen con el nombre de atributos de calidad, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios.

4.3 Cualidades por la que puede ser evaluada una arquitectura.

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

- Seguridad: Es la medida de la habilidad del sistema de resistirse al uso no autorizado y negar los servicios, mientras los provee a usuarios legítimos.
- Modificabilidad: Es la habilidad de realizar cambios al sistema en forma rápida y a bajo costo.

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

4.4 Métodos de evaluación de arquitecturas.

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

De acuerdo con Kazman (22) el método ARID es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura. Según los autores, ARID es un híbrido entre Active Design Review (ADR) y Architecture Trade-Off Method (ATAM), descrito anteriormente.

Según el planteamiento anterior el método ATAM (Método de Análisis de Acuerdos de Arquitectura) surge, del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos

de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros; esto es, los tipos de acuerdos que se establecen entre ellos.

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

Para evaluar la arquitectura mediante el método de ARID se deben tener en cuenta los atributos de calidad contemplados en la convivencia del diseño evaluado, ya que la etapa del proyecto en la que se aplica es a lo largo del diseño de la arquitectura.

4.5 Evaluación de la arquitectura del sistema.

El método ARID surge de combinar las mejores cualidades de los métodos ADRs y los métodos basados en escenarios. De los ADRs nos quedamos con la participación activa de los entrevistados. Del ATAM nos quedamos con la idea de la generación de los escenarios para demostrar que el diseño propuesto cumple con los requisitos.

La arquitectura propuesta en este trabajo se encuentra representada a un alto nivel (diseño poco detallado), esto trajo consigo la dificultad de seleccionar escenarios concretos (como por ejemplo: validación automática de la entrada de datos) que permitiesen validar si la arquitectura responde a requisitos específicos. Debido a ello, se decidió validar la arquitectura evaluando como las vistas seleccionadas responden a cada uno de los principales requerimientos y restricciones del sistema, constituyendo nuestros escenarios los siguientes:

Rendimiento del sistema: Se debe consultar la vista de comunicación de procesos para analizar la concurrencia del sistema. También se debe consultar la vista de despliegue.

Modificabilidad y evolución del sistema: Para analizar el impacto de un cambio, las vistas de uso y de descomposición son muy útiles. También la vista de capas muestra como un cambio en una capa mas baja puede ser ocultado detrás de sus interfaces y no impactara en las capas superiores.

Seguridad del sistema: La vista de despliegue es usada para ver los puntos de contacto del sistema con el exterior, así como el software encargado de establecer la interacción; también muestra donde los aspectos autenticación e integridad son manejados.

Disponibilidad del sistema: Una vista de comunicación de procesos puede ayudar a analizar los procesos de mayor concurrencia. La vista de despliegue es usada para mostrar posibles puntos de fallo. Pueden definirse números de fiabilidad para un modulo como prioridad en la vista de descomposición.

Portabilidad del sistema: El sistema será desarrollado utilizando PHP – Apache - PostgreSQL, combinación que es compatible con los sistemas operativos mas populares: Linux, Windows, FreeBSD y Mac OS.

Independencia entre la aplicación y los sistemas gestores de bases de datos: Esto se logra con la capa de persistencia en la vista de datos. Esta capa media entre la capa de modelo del dominio y la capa de fuentes de datos, permitiendo que para la primera sea transparente la manera en que la información es persistida o almacenada en la fuente de datos. La capa de persistencia además tendrá como función fundamental, lograr que el sistema funcione correctamente independiente de gestor de bases de datos seleccionado.

4.6 Conclusiones.

Después de realizarle una evaluación según los atributos de calidad, haciendo necesarios que permita el funcionamiento de los requisitos del cliente durante el desarrollo del Sistema, se puede obtener una aplicación robusta y permisible a los cambios futuros.

CONCLUSIONES

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

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

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

RECOMENDACIONES

- El refinamiento constante de la arquitectura propuesta, durante el ciclo de desarrollo.
- La revisión de cada uno de los escenarios de la aplicación para evaluar a fondo las restricciones que se le imponen a la arquitectura.
- El análisis de las dependencias entre los subsistemas de implementación para evitar que existan cuellos de botella en alguno de ellos y que se vea afectado en cuanto a tiempo la entrega de cada una de las actividades de implementación.
- Ir incorporando los diferentes patrones que se vayan utilizando a medida que se desarrolle la construcción del sistema, en el documento de la arquitectura.
- La reutilizar la arquitectura de la primera versión de SIGAC a otros sistemas con características similares.

REFERENCIAS BIBLIOGRÁFICAS.

1. **Pedraza Rodríguez, Lina.** *RESOLUCION No. 340/03 .* La Habana : s.n., 2003.

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

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

4. **Perry, Dewayne and Wolf, Alexander.** *Foundations for the study of software architecture.* s.l. : ACM SIGSOFT Software Engineering Notes, 1992.

5. **Shaw, Mary and Garlan, David.** *Software Architecture. Perspectives of an Emerging Discipline.* s.l. : Prentice Hall, 1996.

6. **Jacopbson, Ivar.** *The Unified Software Development Process.* s.l. : Addison Wesley, 1999.

7. **Pressman, Roges S.** *Ingenieria de Software un Enfoque Práctico.* Ciudad de la Habana : Felix Varela, 2005.

8. **Kruchten, Philippe.** *The 4+1 View Model of Architecture.* s.l. : IEEE Software, 1995.

9. **Booch, Grady, Rumbaugh, James and Jacobson, Ivar.** *El Lenguaje Unificado de Modelado.* Madrid : Addison-Wesley, 1999.

10. **Shaw, Mary.** *Pattern Languages of Program.* s.l. : Addison-Wesley, 1996.

11. **Monroe, Robert, et al.** Stylized architecture, design patterns, and objects. [Online] http://citeseer.nj.nec.com/monroe96stylized.html.

12. **Monroe, Robert, et al.** *Architectural Styles, design patterns, and objects.* s.l. : IEEE Software, 1997.

13. **White, Sharon and Lemus-Olalde, Cuauhtémoc.** The software architecture process. [Online] 1997. http://nas.cl.uh.edu/whites/webpapers.dir/ETCE97pap.pdf.

14. **Garlan, David and Shaw, Mary.** *An introduction to software architecture.* s.l. : CMU Software Engineering Institute Technical Report, 1994. CMU/SEI-94-TR-21.

15. **Jacobson, Ivar, Booch, Grady and Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* Madrid : PEARSON EDUCACION, 2000.

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

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

18. **BIBEAULT, BEAR and KATZ, YEHUDA.** *jQuery in Action.* Greenwich : Manning Publications Co., 2008. 1-933988-35-5.

19. **Potencier, Fabien and Zaninotto, François.** Symfony, la guía definitiva. *Librosweb.es.* [Online] 2007. http://www.librosweb.es/symfony.

20. **Sweat, Jason E.** *Guide to PHP Design Patterns.* Toronto : Marco Tabini & Associates, Inc., 2005. 0-9735898-2-5.

21. **Welicki, L.** *Patrones y Antipatrones: una Introducción.* Madrid : Anaya, 2007.

22. **Kazman, Rick, Clements, Paul and Klein, Mark.** *Evaluating Software Architectures: Methods and Case Studies.* s.l. : Addison Wesley, 2004.

BIBLIOGRAFÍA.

Schmidt, Douglas, et al. *Pattern Oriented Software Architecture. Volumen 1 y 2.* s.l. : Addison Wesley.

Bass, L, Clements, P and Kazman, R. *Software Architecture in Practice.* s.l. : Addison Wesley, 1998.

Booch, Grady. *Design & Use of Software Architectures.* s.l. : Addison Wesley, 2000.

Kruchten, P. *The 4+1 View Model of Architecture.* s.l. : IEEE Software, 1995.

Len Bass, P and Kazman, Rick. *Software Architecture in Practice (2nd Ed.).* s.l. : Addison Wesley, 2003.

Shaw, Mary. *Software Architecture. Perspectives on an Emerging Discipline.* s.l. : Pentice Hall, 1996.

Monroe, Robert, Melton, Ralph and Garlan, David. *Architectural Styles, design patterns, and objects.* s.l. : IEEE Software, 1997.

Rumbaugh, James. *The Unified Modeling Language Reference Manual.* s.l. : Addison Wesley, 1999.

Gutmans, Andi. *PHP 5 Power programming.* Indianapolis : Prentice Hall, 2005.

Douglas, Korry and Douglas, Susan. *PostgreSQL. The comprensive guide to building, programming, and administering PostgreSQL databases.* s.l. : Sams Publishing, 2006.

Shklar, Leon and Rosen, Richard. *Web application architecture : principles, protocols, and practices.* New Jersey : Jhon Wiley & Sons, Inc, 2003.

Clements, Paul. *Documenting sotfware architecture : views and beyond.* s.l. : Pearson Education, 2002.

BIBEAULT, BEAR and KATZ, YEHUDA. *jQuery in Action.* Greenwich : Manning Publications Co., 2008. 1-933988-35-5.

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

Garlan, David and Shaw, Mary. *An introduction to software architecture.* s.l. : CMU Software Engineering Institute Technical Report, 1994. CMU/SEI-94-TR-21.

Jacobson, Ivar, Booch, Grady and Rumbaugh, James. *El Proceso Unificado de Desarrollo de Software.* Madrid : PEARSON EDUCACION, 2000.

Kazman, Rick, Clements, Paul and Klein, Mark. *Evaluating Software Architectures: Methods and Case Studies.* s.l. : Addison Wesley, 2004.

Potencier, Fabien and Zaninotto, François. Symfony, la guía definitiva. *Librosweb.es.* [Online] 2007. http://www.librosweb.es/symfony.

Sweat, Jason E. *Guide to PHP Design Patterns.* Toronto : Marco Tabini & Associates, Inc., 2005. 0-9735898-2-5.

Welicki, L. *Patrones y Antipatrones: una Introducción.* Madrid : Anaya, 2007.

Blanco, Santiago. *Utilizacion de UML como un ADL para documentacion de Arquitecturas*. Espacio de Regional Architect Forum. [Online] 2006. http://raf06-pta-del-este.spaces.live.com/blog/cns!E1995E17D66AB9C!225.entry

Autores Varios. *Ingenieria de Software 1*. [Online] 2008. http://ing.de.soft1.googlepages.com/home

Molpeceres, Alberto. *Procesos de desarrollo: RUP, XP y FDD*. 2002

ANEXOS

Anexo 1. Estándares de codificación.

Los estándares de código son importantes y de obligatorio cumplimiento para los proyectos en nuestra Universidad, ya que los equipos de desarrollo se desintegran con el tiempo (estudiantes que se gradúan y se van de la universidad) y los sistemas deben ser continuados y mantenidos por equipos de soporte, que necesitan entender el código para posibles cambios.

En nuestro proyecto se utilizarán estándares internacionalmente aceptados, aunque adaptados a la estructura y necesidades del proyecto.

Estructuras de control

Aquí se incluye todas las conocidas: if, for, while, switch, etc. Veamos un ejemplo del if –elseif – else y uno para el switch:

```
if ((condicion1) || (condicion2))
{
    accion1;
}
elseif ((condicion3) && (condicion4))
{
    accion2;
}
else
{
    default;
}

switch (condition)
{
case 1:
    accion1;
    break;

case 2:
    accion2;
    break;

default:
    defaultaccion;
    break;
}
```

Llamadas a Funciones

Las funciones se llamaran sin espacio entre el nombre de la función, el paréntesis de apertura y el primer parámetro; espacio entre comas y cada parámetro, y sin espacio entre el ultimo parámetro, el paréntesis de cierre, y el punto y coma. Ejemplo:

```
$var = foo($bar, $baz, $quux);
```

En caso de bloque de funciones se alinearan a la altura de la función con nombre mas largo:

```
$short            =     foo($var);
$long_function    =     foo($var);
```

Comentarios

Se utilizara el formato de phpDocumentor. Se usaran para documentar el proyecto. Para los comentarios de una sola línea se utilizara el // y para comentario de varias línea /* */.

Ejemplo:

```
// este es un comentario de una línea
/*
*comentario
*de
*varias
*líneas
*/
```

Para los cometarios de inicio de fichero se utilizara el estándar aprobado por PEAR y utilizado por el phpDocumentor. Este cometario tendrá la siguiente estructura:

```
<?php

/**
* Short description for file
*
* PHP versions 4 and 5
*
* @category   CategoryName
* @package    PackageName
* @author     Original Author <author@example.com>
* @author     Another Author <another@example.com>
* @copyright  1997-2005 The PHP Group
* @license    http://www.php.net/license/3_0.txt  PHP License 3.0
* @version    CVS: $Id:$
*/
class foo
```

```
{
}

?>
```

Short description: se realizara una breve descripción del documento, cual es su función y datos de interés.

PHP versions: se indicara para que versión de PHP es valido el código.

@**category** Categoría a la que pertenece
@**package** Nombre del paquete al que pertenece
@**author** Nombre del autor <author@example.com>
@**author** Nombre co-autor <another@example.com>
@**copyright** Año o periodo Quien lo licencia
@**license** Se pondrá la licencia bajo la cual se libera el código (si lo amerita). Si son más de 1 se pondrán una bajo la otra
@**version** versión del archivo

En el caso de las funciones, se utilizara el estándar del phpDocumentor:

```
/**
   * Enter description here...
   *
   * @param string $var
   * @param string $var1
   * @return bool
   */
public function Mifunction($var, $var1)
  {
        return true;
  }
```

Enter description here: una descripción del método

@**param:** listado de parámetros de entrada. Se debe utilizar de la forma @**param tipo nombre_parametro**

@**return:** Tipo de la salida (en caso de haberla). Su forma es @**return tipo**

PHP tags

Siempre usar **<?php ?>** para delimitar código PHP. Esto es requerido por la sintaxis de Symfony.

Formato de los ficheros

Todos deben:

- Ser guardados como texto ASCII
- Usar la codificación ISO-8859-1
- Tener una línea en blanco después de la última línea de código.

Tratamiento de errores

Se utilizara el estándar try – throw - catch. En casos de errores de validaciones o errores simples no es necesario poner todo el código. Ejemplo:

```
function divide($x, $y)
{
    if ($y == 0)
    {
        throw new Exception('Division by zero');
    }
}
```

En caso contrario se utilizara toda la estructura conocida. Esta debe usarse en código donde pueda haber salidas inesperadas o rupturas del código.

```
function preTaxPrice($retailPrice, $taxRate)
{
    try
    {
        //código sospechoso
    }
    catch (Exception e)
    {
        //tratamiento del error
    }
}
```

Funciones

En el proyecto se utilizara el framework Symfony. En Symfony las acciones son métodos con el nombre **executeNombreAccion**, de la clase **nombreModuloActions**. Estos se encuentran en el archivo **actions.class.php** en el directorio **actions/** del modulo

```
public function executeMiAccion()
{

}
```

Por defecto todas las acciones se declararan e implementaran en dicho archivo, pero puede ser conveniente separarlos en varios archivos. En este caso se crearía un archivo por acción. Cada archivo se llamaría **nombreAccionAction.class.php**. La clase se llamaría entonces **nombreAccionActions** y heredaría de **sfAction** en vez de **sfActions** cuando se realiza en un solo archivo. Aquí los métodos se llamarían simplemente **execute**. ¿Como quedaría esto? Veamos el ejemplo:

Archivo actions.class.php

```
class mimoduloActions extends sfActions
{
public function executeIndex()
{}
public function executeList()
{}
}
```

Si se separa en varios archives quedaría en la siguiente forma:

Archivo indexAction.class.php

```
class indexActions extends sfAction
{
public function execute ()
{}
}
```

Archivo listActions.class.php

```
class listActions extends sfAction
{
```

```
public function execute ()
{}
}
```

Para que una acción se realice al principio de cada acción o método, se creara una acción llamada **preExecute ()** con el código que se quiera. Una acción llamada **postExecute ()** contendrá código que será ejecutado al final de todas las acciones.

Para crear métodos propios se recomienda que sean privados o protegidos y se pueden hacer siempre que no lleven la palabra execute en el nombre. Se seguirá el estilo **"BSD/Allman"**:

```
protected function fooFunction($arg1, $arg2 = '')
{
    if (condition)
    {
        //proceso;
    }
    return $val;
}
```

Argumentos con valores por defecto se pondrán al final de la lista de argumentos. Ejemplo:

```
protected function connect($dsn, $persistent = false)
{
    if (is_array($dsn))
    {
        $dsninfo = &$dsn;
    } else
    {
        $dsninfo = DB::parseDSN($dsn);
    }

    if ((!$dsninfo) || (!$dsninfo['phptype']))
    {
        return $this->raiseError();
    }

    return true;
}
```

Ejemplo:

Archivo actions.class.php

```
class mimoduloActions extends sfActions
{
public function preExecute()
```

```
{
/* El código insertado aquí se ejecuta al principio de cada llamada a una acción*/
}
public function executeIndex()
{}
public function executeList()
{
$this->miPropioMetodo(); // Se puede acceder a cualquier método de la clase acción
}
public function postExecute()
{
// El código insertado aquí se ejecuta al final de cada llamada a la acción
}
protected function miPropioMetodo()
{
// Se pueden crear métodos propios, siempre que su nombre no comience por
"execute". En ese caso, es mejor declarar los métodos como protected o prívate.
}
}
```

Variables

La declaración de las variables seguirá la siguiente estructura:

- Los nombres deben ser cortos y descriptivos.
- Serán con letra minúscula.
- Si el nombre lleva más de una palabra, estas se separaran por el símbolo “_”.
- Si son atributos de una clase siempre debe tener la visibilidad de la misma (privado, público, protegido).
- Las variables de tipo **string** se llamaran txt_nombre.
- Las variables de tipo **entero** se llamaran int_nombre.
- Las variables de tipo **float** se llamaran dec_nombre.
- Las variables de tipo **boolean** se llamaran bin_nombre.
- Las variables de tipo **array** se llamaran arr_nombre.
- Las variables de tipo **objeto** se llamaran obj_nombre.
- Las variables de tipo **char** se llamaran car_nombre.

Clases

En caso de necesitarse la declaración de alguna clase, se seguirá la siguiente estructura:

- Los nombres deben ser cortos y descriptivos.
- Serán en minúsculas con letra inicial mayúscula.

Si el nombre lleva más de una palabra, éstas no se separarán y las palabras seguirán la regla 2 (notación UpperCamelCase).

Anexo 2. Estándares para la bases de datos

Las bases de datos seguirán la siguiente regla de nombre:

MAC_<Nombre>. Ej: MAC_Recepcion

1. Los nombres serán cortos y descriptivos.
2. Serán en minúsculas con letra inicial mayúscula.
3. Si el nombre lleva más de una palabra, éstas se separarán por el símbolo "_".

Tablas

Siguen la siguiente regla de nombre:

<Nombre>

El nombre siempre será en singular. Además:

a. Se antepondrá una "n" para las tablas de nomencladores. Ej: nProvincia

b. Se antepondrá una "d" para las tablas de datos. Ej: dPersona

c. Se antepondrá una "r" para las tablas que representen relaciones. Ej: rPersona_Estado. El nombre además consistirá en los nombres de las entidades que se relacionan, unidas por "_".

d. Será corto y descriptivo.

e. Será en minúscula con letra inicial mayúscula.

Para los campos de las tablas:

- Para carnet de identidad se usará: CI.

- Para campos que sean booleanos se antepondrá un "Es_" delante del nombre, por ejemplo: "Es_Directivo".

- Los campos identificadores que sean del tipo "Id" se nombrarán así mismo "id".

-Los nombres deben ser cortos y descriptivos.

-Serán en minúscula con letra inicial mayúscula.

-Si el nombre lleva más de una palabra, estas se separaran por el símbolo "_" y seguirán la regla anterior.

Vistas

Siguen la siguiente regla de nombre:

Vs_<Nombre>. Ej: Vs_DatosPersona

Los nombres deben ser lo mas claros posibles, es decir que brinden la mayor información posible sobre la vista.

Procedimientos Almacenados o Funciones

Func_<Entidad>_<Operacion>

Entidad: Es el nombre de la Entidad principal sobre la cual actúa el procedimiento.

Operación: Indica la acción que efectúa el procedimiento y puede tomar los siguientes valores:

- Ins para inserción

- Elm para eliminación

- Act para actualizar

- Sel para selección

Ejemplos:

- Func_Persona_Ins

- Func_Provincia_Sel

Nomencladores Globales

Información de personas

Nombre Del Campo	Explicación	Posibles valores.
Id	Identificador único de las personas.	Cadena
Nombre	Nombre de la persona, ambos nombres si es compuesto. No incluye apellidos.	Cadena
Primer_Nombre	Primer nombre de una persona, en caso de nombres compuestos.	Cadena
Segundo_Nombre	Segundo nombre de una persona, en caso de nombre compuesto.	Cadena
Primer_Apellido	Primer apellido de la persona	Cadena
Segundo_Apellido	Segundo apellido de la persona	Cadena
CI	Carnet de Identidad. Este campo no deberá usarse como llave pues puede repetirse en ciertas ocasiones. Sin embargo, se recomienda su uso para hacer búsquedas.	Cadena
Sexo	Sexo de la persona	Caracter. Posibles valores: M, F.

Información de direcciones y delegaciones

Nombre Del Campo	Explicación	Posibles valores.
Id	Identificador de áreas.	Numérico
Nombre_Area	Nombre del área	Cadena

Información de municipios

Nombre Del Campo	Explicación	Posibles valores.
Id	Identificador de los municipios del país.	Numérico
Nombre_Municipio	Nombre del Municipio	Cadena

Información de provincias

Nombre Del Campo	Explicación	Posibles valores.
Id	Identificador de las provincias del país.	Numérico
Nombre_Provincia	Nombre de la provincia	Cadena

Anexo 3. Estándares para el diseño.

Formularios

Los formularios se llamaran o crearan utilizando helpers.

La sentencia de los elementos de formulario más comunes son (para más información manual de Symfony cap. 10 pág. 261):

form: form_tag(). Se le pasa como parámetro la acción que lo atenderá y como segundo parámetro opciones como el método (get - post), la clase css que le dará estilo, etc.

```
<?php echo form_tag('prueba/guardar', 'method=get multipart=true
class=formularioSimple')
?></form>
```

Campo de texto: input_tag(). Se le pasa como parámetro el nombre del mismo, el valor inicial y parámetros adicionales validos. Todos deben llamarse txtnombre.

```
<?php echo input_tag('txtnombre', 'valor inicial', 'maxlength=20') ?>
```

Campo de texto grande: textarea_tag(). Idéntico al anterior. Todos deben llamarse textnombre.

```
<?php echo textarea_tag('textnombre', 'valor inicial', 'size=10x20') ?>
```

Checkbox: checkbox_tag(). Se le pasa como parámetro el nombre, el valor, y si esta activado o no. Todos deben llamarse checkname.

```
<?php echo checkbox_tag('checksoltero', 1, true) ?>
```

Radio Button: radiobutton_tag(). Sus parámetros: nombre, valor, activado o no. Todos deben llamarse radioname.

```
<?php echo radiobutton_tag('radioestado[]', 'valor1', true) ?>
<?php echo radiobutton_tag('radioestado[]', 'valor2', false) ?>
```

Listas desplegables: select_tag(). Sus parámetros nombre, listado de opciones que pueden ser array asociativos o enumerativo y la opción seleccionada. Todos deben llamarse selectname.

```
<?php echo select_tag('selectpago', options_for_select(array(
'Visa',
'Eurocard',
'Mastercard'
), 0)) ?>
```

```
<?php echo select_tag('selectnombre', options_for_select(array(
'Steve' => 'Steve',
```

```
'Bob' => 'Bob',
'Albert' => 'Albert',
'Ian' => 'Ian',
'Buck' => 'Buck'
), 'Ian')) ?>
```

Lista desplegable que permite una selección múltiple(los valores seleccionados se pueden indicar en forma de array).

```
<?php echo select_tag('selectpago', options_for_select(
array('Visa' => 'Visa', 'Eurocard' => 'Eurocard', 'Mastercard' => 'Mastercard'),
array('Visa', 'Mastecard')
), 'multiple=multiple') ?>
```

Campo para archivos: input_file_tag(). Se le pasa el nombre y parámetros extras validos. Todos deben llamarse filename.

```
<?php echo input_file_tag('filenombre') ?>
```

Contraseña: input_password_tag(). Se le pasa nombre, valor, adicionales validos. Todos deben llamarse passname.

```
<?php echo input_password_tag('passnombre', 'valor') ?>
```

Campo oculto: input_hidden_tag(). Idéntico al campo de contraseña. Todos deben llamarse hidename.

```
<?php echo input_hidden_tag('hidenombre', 'valor') ?>
```

Submit: submit_tag(). Se le pasa el texto que mostrara el botón.

```
<?php echo submit_tag('Guardar') ?>
```

Submit con imagen: submit_image_tag (). Se le pasa como parámetro el nombre de la imagen que mostrara. Dicha imagen debe estar en la carpeta **web/images/** de la instalación del Symfony.

```
<?php echo submit_image_tag('imagen_envio') ?>
```

Campos de fecha: input_date_tag(). Se le pasa el nombre, el formato de fecha y si se muestra un calendario interactivo. Todos deben llamarse datename.

```
<?php echo input_date_tag('datefechanacimiento', '2005-05-03', 'rich=true') ?>
```

Los formatos de fecha deben ser formatos validos tales como

año – mes – día (2007-11-27) y día mes año (27 October 2007)

Contenedores y CSS

Los **div** o contenedores deben llamarse **contectparte**, donde la parte es lo que contiene. Por ejemplo contectmenu, contectbody, contectleft.

Los nombres de las tablas deben ser lo mas explicitas posible. Deben llamarse **tablename**. Ejemplo tableusuarios, tableprocesos.

Las clases en el ccs deben tener nombres lo mas explícitos posible y siempre que no se refieran a un objeto en especifico se nombraran **nameclass**.

GLOSARIO DE TÉRMINOS

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

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

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

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

Broadcast: es un modo de transmisión de información donde un nodo emisor envía información a una multitud de nodos receptores de manera simultánea, sin necesidad de reproducir la misma transmisión nodo por nodo.

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

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

Cliente: es un ordenador que accede a recursos y servicios brindados por otro llamado Servidor, generalmente en forma remota.

Componentes: más conocido como, los componente de Software, que son todo aquel recurso desarrollado para un fin concreto y que puede formar solo o junto con otro/s, un entorno funcional

requerido por cualquier proceso predefinido. Son independientes entre ellos, y tienen su propia estructura e implementación. Si fueran propensos a la degradación debieran diseñarse con métodos internos propios de actualización.

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

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

Estereotipo: Define un nuevo significado de la semántica para el elemento a modelar.

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

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

Hardware (soporte físico): se utiliza generalmente para describir los artefactos físicos de una tecnología. Es el conjunto de elementos físicos que componen una computadora Disco Duro, CD-ROM, etc.

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

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

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

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

IDE (Integrated Development Environment): Un entorno de desarrollo integrado, es un programa compuesto por un conjunto de herramientas para un programador. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios.

IEEE: corresponde a las siglas de The Institute of Electrical and Electronics Engineers, el Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas. Es la mayor asociación internacional sin fines de lucro formada por profesionales de las nuevas tecnologías, como ingenieros eléctricos, ingenieros en electrónica, científicos de la computación e ingenieros en telecomunicación.

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

Informática: es la disciplina que estudia el tratamiento automático de la información utilizando dispositivos electrónicos y sistemas computacionales. Es la unión sinérgica del cómputo y las comunicaciones.

Ingeniería del Software: es el estudio de los principios y metodología para desarrollo mantenimiento de de sistema de software.

ISO: la Organización Internacional para la Estandarización o International Organization for Standarization (ISO), es una organización internacional no gubernamental, compuesta por representantes de los organismos de normalización (ONs) nacionales, que produce normas internacionales industriales y comerciales. Dichas normas se conocen como normas ISO

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

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

Mapeo objeto-relacional: (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos.

MOF: metalenguaje utilizado para extender las funcionalidades de UML.

OCL: lenguaje hibrido orientado a objetos-funcional, tipificado. Lenguaje propuesto para especificar las restricciones semánticas del diagrama de clases de UML

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

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

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

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

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

POSA: **P**attern-**O**riented **S**oftware **A**rchitecture

Protocolo: Conjunto de normas y procedimientos útiles para la transmisión de datos, conocido por el emisor y el receptor.

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

RAM (**R**andom **A**ccess **M**emory): memoria de acceso aleatorio ó memoria de acceso directo.

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

SEI (Software Engineering Institute): es un instituto federal estadounidense de investigación y desarrollo, fundado por el Congreso Estadounidense en 1984 para desarrollar modelos de evaluación y mejora en el desarrollo de software, que dieran respuesta a los problemas que generaba al ejército estadounidense la programación e integración de los sub-sistemas de software en la construcción de complejos sistemas militares. Financiado por el Departamento de Defensa estadounidense y administrado por la Universidad Carnegie Mellon.

Servidor: Una aplicación informática o programa que realiza algunas tareas en beneficio de otras aplicaciones llamadas clientes.

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

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

Servicio WEB (en inglés Web service): es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores.

Sistema Operativo: es un conjunto de programas destinados a permitir la comunicación del usuario con un computador y gestionar sus recursos de una forma eficaz.

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

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

Software (soporte lógico): los componentes intangibles de una computadora, es decir, al conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica.

TCP/IP: es un conjunto de protocolos de red que permiten la transmisión de datos entre redes de computadoras.

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

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

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

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

WWW (World Wide Web): es un sistema de documentos de hipertexto enlazados y accesibles a través de Internet. Con un navegador Web, un usuario visualiza páginas Web que pueden contener texto, imágenes u otros contenidos multimedia.

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