Universidad de las Ciencias Informáticas
Facultad 10

Trabajo de Diploma para optar por el título
de Ingeniero en Ciencias Informáticas.

Título: Descripción de la arquitectura de software de la herramienta de gestión de proyectos dotProject

Autor: Wilde Manuel Matos Lezcano.

Tutores: Ing. Yaumara Arce Ajo
: Ing. Geykel Díaz Azcon

Ciudad de La Habana, Cuba, 2009.
"Año del 50 Aniversario del triunfo de la Revolución"

"Si fuéramos capaces de unirnos...Que hermoso y qué cercano sería el futuro!"

Ernesto Ché Guevara

Declaración de Autoría

Declaro que soy 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 _____.

_____

Wilde Manuel Matos Lezcano

_____ _____

Ing. Yaumara Arce Ajo Ing. Geykel Díaz Azcon

Agradecimientos

A mis dos lindos y maravillosos padres, por haber estado a mi lado siempre, por apoyarme en todo, por ayudarme a levantarme cuando me caía, por sus consejos, su amor, dedicación, cariño, respecto, por haber hecho de mí el hombre que soy. A ellos hoy y siempre, mil gracias.

A mis tutores Yaumara y Geykel por su ayuda, empeño y dedicación.

A mi tía Vicenta, que siempre me ha apoyado, ayudado y querido como si fuese su hijo, a quien le voy a estar eternamente agradecido por ser una madre más para mí.

A mi hermano Carlos Enrique por su apoyo incondicional.

A mi novia Rosalia, por estar a mi lado y apoyarme en todo momento, por su incondicionalidad y comprensión, por darme su amor y cariño cuando más lo necesitaba y por confiar siempre en mí.

Al resto de mi familia por apoyarme en todo momento cuando más los necesité.

A mi amigo Karel Tito, por su apoyo incondicional, en todo momento, por soportarme durante el desarrollo de este trabajo y por su ayuda en la elaboración del mismo.

Agradecimientos

A mi amigo y jefe de proyecto Yoandy Pérez Villazón, por su dedicación y apoyo en la investigación realizada.

A mis amigos y compañeros de aula, a los que quedaron en el camino que aún están presentes en mi memoria y los que aún están para apoyarme en todo momento, así como a todos aquellos que han estado a mi lado de una forma u otra cuando los he necesitado.

A mis compañeros del grupo de proyectos "Unicornios", a los cuales considero mis hermanos.

A mis compañeros del CDI "MySantha", en Venezuela, los que fueron mi familia durante 5 meses, y me apoyaron en todo momento.

Muchas gracias.

Dedicatoria

A la memoria de mi abuela Rosa, que aunque ya no este con nosotros, siempre me aconsejó y guió por el camino correcto, y fue una fuente de inspiración en estos 5 años de estudio.

A mi madre querida, por creer siempre en mí, por su compañía y apoyo, por ser siempre más que una madre una amiga, por sus consejos, su paciencia y su amor. Por ser la mejor madre del mundo.

A mi padre, mil gracias por su dedicación y sus consejos, por apoyarme a pesar de la distancia, y confiar siempre en mí, por ser más que un padre, mi mejor amigo.

A alguien que gracias a esta universidad conocí, a ti Rosy, por darme lo mejor de ti en todo momento, por siempre estar a mi lado.

Al resto de mi familia por confiar siempre en mí.

Resumen

En este trabajo se describe la arquitectura de la herramienta gestora de proyectos dotProject, para lograr este objetivo es necesario realizar una sistematización sobre las distintas arquitecturas existentes, lo cual nos permitirá determinar cual es la arquitectura que utiliza dicha herramienta de acuerdo a sus características. Para el desarrollo del mismo se usarán herramientas que tributen al Software Libre, ya que el mismo tiene la posibilidad de ser modificado por el usuario a su conveniencia y sus necesidades.

En el trabajo se hace un estudio de los conceptos fundamentales sobre la arquitectura, los estilos y patrones arquitectónicos. Además se plantean los tipos de arquitecturas más comunes, las herramientas y el lenguaje de modelado utilizados para el desarrollo del mismo. se recoge también la descripción del modelo de dominio del dotProject, la estructura de directorios de dicha herramienta, los distintos métodos y funciones, una guía para crear nuevos módulos, la descripción de la arquitectura y una descripción de la base de datos de la herramienta en cuestión.

Actualmente la documentación sobre la arquitectura del dotProject no existe o no esta disponible para los usuarios, es por ello que la realización de este trabajo de diploma tiene un impacto significativo, ya que servirá de guía a todas aquellas personas que deseen desarrollar nuevos módulos o reimplementar funcionalidades ya existentes.

Índice

Índice

Introducción

La gestión de proyectos es una rama especializada de la Ingeniería de Software que se puede describir como un proceso de planteamiento, ejecución y control de un proyecto, desde su comienzo hasta su conclusión, con el propósito de alcanzar un objetivo final en un plazo de tiempo determinado, con un coste y nivel de calidad determinados, a través de la movilización de recursos técnicos, financieros y humanos. Incorporando variadas áreas del conocimiento, su objetivo final es el de obtener el mejor resultado posible del trinomio coste-plazo-calidad.

La Universidad de las Ciencias Informáticas (UCI), es un centro donde actualmente se lleva a cabo una gran producción de software, tanto nacionales como de exportación. Pero el proceso de gestión en algunos de los proyectos productivos en esta institución no se realiza de manera adecuada, lo cual provoca resultados insatisfactorios tanto para nuestra universidad como para los clientes, pues no se pueden brindar fechas de terminación exactas y además provoca incumplimiento con las fechas de entrega establecidas, afecta la toma de decisiones en el proyecto y la calidad de los entregables. Estos problemas se pueden solucionar utilizado una herramienta gestora de proyecto, para ello en investigaciones anteriores se propuso la herramienta dotProject para emplearla en la facultad 10, específicamente en el grupo de proyectos “Unicornios”, en el cual se utiliza hace dos años y se han obtenido resultados satisfactorios, pero esta herramienta no posee una arquitectura definida o la información de dicha arquitectura no está disponible en la red, por lo que nuestra **situación problémica** es la inexistencia de la documentación sobre la arquitectura de software de la herramienta gestora de proyectos dotProject, lo cual le permitiría a usuarios y desarrolladores guiarse para crear nuevos módulos o darle algún tipo de soporte a dicha herramienta.

Como consecuencia de la situación problémica antes expuesta se deriva el siguiente **problema científico de la investigación**: ¿Cómo describir la arquitectura de software de la herramienta de gestión de proyectos dotProject que sirva de guía a desarrolladores y usuarios para crearle nuevos módulos y darle soporte aesta herramienta?

Introducción

Para describir la arquitectura de software del dotProject, es necesario realizar una sistematización sobre las distintas arquitecturas existentes, para determinar de acuerdo a las características del sistema en cuestión cual es la utilizada. Por lo que nuestro **objeto de estudio** son las arquitecturas de software, y nuestro **campo de acción** se centra en la arquitectura del sistema de gestión de proyectos dotProject.

Esta investigación tiene como **objetivo general** identificar y describir la arquitectura de software de la herramienta de gestión de proyectos dotProject. Para darle cumplimiento a dicho objetivo general se trazaron los siguientes **objetivos específicos**:

- Sistematizar sobre los diferentes tipos de arquitectura de software existentes.
- Sistematizar sobre las características, el funcionamiento y la arquitectura del dotProject.
- Elaborar un documento que describa la arquitectura de dicha herramienta de gestión de proyectos, así como los pasos a seguir para la creación de un módulo.

Las tareas de investigación a cumplir para desarrollar cada uno de los objetivos son:

- Estudiar las diferentes arquitecturas de software existentes.
- Sistematizar sobre el funcionamiento del dotProject.
- Describir la estructura de directorios del dotProject y su arquitectura.
- Desarrollar un módulo que permita generar nuevos módulos.

Para el cumplimiento de estas tareas se han utilizado Métodos Científicos, mediante el **Método Teórico Analítico-Sintético** ya que se analizaron documentos relacionados con el tema, que brindan gran cantidad de información para la realización de nuestro trabajo, obteniendo los elementos más importantes de manera que se puedan elaborar conclusiones; se utiliza además el **Análisis Histórico-Lógico** para el estudio de la evolución y características de las arquitecturas existentes. En general, a través de estos métodos pudimos obtener toda la información necesaria para desarrollar nuestro trabajo.

Posibles resultados:

Se obtendrá una descripción completa de la arquitectura de la herramienta de gestión de proyectos dotProject, lo cual facilitará gran cantidad de información a los que deseen desarrollar nuevos módulos, reimplementar o darle soporte a esta herramienta.

El presente trabajo consta de una Introducción, 2 capítulos, las conclusiones generales, recomendaciones, referencias bibliográficas y bibliografía utilizada durante el desarrollo del trabajo y por último los anexos que complementan el cuerpo del trabajo.

El primer capítulo contempla la fundamentación teórica del tema, donde se define el marco teórico y el modelo teórico de la investigación, el estudio del estado del arte de los principales conceptos abordados durante la investigación, así como las distintas herramientas a utilizar en el desarrollo del trabajo. El capítulo dos presenta de forma general la explicación del modelo de dominio, la organización de los diferentes directorios del dotProject con una pequeña descripción del contenido de los mismos, se plantea una guía para la creación de nuevos módulos, se realiza la propuesta de un módulo que permitirá generar nuevos módulos, así como la descripción de la arquitectura y de la base de datos del dotProject.

1 Capítulo 1. Fundamentación Teórica

En este capítulo se analizan los aspectos fundamentales a tener en cuenta para el desarrollo de la descripción de la arquitectura de software de la herramienta de gestión de proyectos dotProject. Además de realizar un análisis de algunas definiciones de arquitectura de software, estilos arquitectónicos y las arquitecturas más comunes existentes. Conjuntamente se hace un análisis de algunas tecnologías y herramientas a utilizar.

1.1 Conceptos generales

A continuación se analizan algunos de los conceptos más significativos a tener en cuenta para llevar a cabo la solución de la descripción de la herramienta dotProject.

1.1.1 Herramienta de gestión de proyectos

Una herramienta gestora de proyectos es aquel software que permite llevar un control minucioso de cada uno de los proyectos de una empresa. El control implica todas las partes del trabajo, como la planificación, desarrollo y producción, así como el trato con el cliente. Este debe ser capaz de gestionar varios trabajos, ya sean internos de la empresa o contratados por clientes. Dentro de cada proyecto o trabajo, debe poderse administrar todo tipo de ítem como:

- **Proyectos**, con descripciones, fechas de entrega, tiempos estimados, etc.
- **Usuarios**, generalmente los trabajadores que están implicados en las tareas de los proyectos.
- **Tareas**, manejando especificaciones, plazos, prioridad, planificaciones de tiempo, etc. y además debe poder asignar recursos de la empresa (generalmente empleados) a cada tarea.

La utilización de las herramientas de gestión de proyectos puede ser algo específico de la persona encargada de la gestión del proyecto (el administrador, gestor o jefe del proyecto). Entonces es un programa que el gestor utiliza y le sirve internamente para saber acerca de la marcha del proyecto (en qué punto se encuentra en cada momento y los tiempos o recursos que necesitará hasta la finalización). Pero también

puede ser una herramienta que utilice todo el equipo de desarrollo y entonces se trata de una utilidad más completa, porque no sólo sirve al gestor, sino también a cada uno de los integrantes de la empresa, para saber las tareas que tienen pendientes y la marcha de cada una de ellas.

Utilizar una herramienta de gestión de proyectos es imprescindible para organizarse, sobre todo en empresas de cierto tamaño, que manejen varios proyectos. Aunque de forma general es muy útil para cualquier grupo de trabajo del tamaño que sea. Sobre todo es muy interesante para modelos de empresa deslocalizada, cada vez más habituales, donde cada persona trabaja de forma remota.(Miguel Ángel Alvarez 2007).

Las herramientas de gestión de proyectos en cuanto a su interfaz se clasifican en:

- **Aplicaciones de escritorios:** son aquellas herramientas que funcionan como una aplicación de escritorio, es decir, un programa como cualquier otro que tengamos en nuestro sistema operativo.
- **Aplicaciones Web:s**on aquellas herramientas creadas con interfaz web, que se acceden en un dominio de Internet o una intranet por medio de un navegador.

1.1.2 Arquitectura de Software

Una arquitectura de software, también denominada arquitectura lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información. Esta aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño.

La Arquitectura de Software establece los fundamentos para que analistas, diseñadores, probadores, programadores, y el resto de los integrantes del equipo o grupo de desarrollo trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades.

Definiciones que existen de la arquitectura de software:

"La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones". (Paul Clemens 2008)

A la hora de realizar una arquitectura hay que tener en cuenta el funcionamiento y la interacción de los componentes, por lo que la estructura debe ser diseñada de acuerdo a los requisitos funcionales y no funcionales, para poder proporcionar una buena confiabilidad, seguridad, integridad, flexibilidad y escalabilidad del sistema.

Al igual que muchos procesos, tanto informáticos como de otra índole, la arquitectura de software ha ido adaptándose constantemente a las necesidades del mercado. Esta evolución se ha dado de manera revolucionaria, generando nuevos paradigmas en la concepción del software. Para saber en qué etapa del proceso está la arquitectura de software, es importante saber primero cómo se define. A grandes rasgos, es una vista del sistema que incluye los componentes principales del mismo, la conducta de éstos según es percibida desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar el objetivo propuesto para el sistema. La definición oficial se ha acordado que sea la que brinda el documento de IEEE Std 1471-2000, donde se declara que: "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". (Rodrigo Escárate 2007)

1.1.3 Patrones arquitectónicos

Un patrón es un modelo que podemos seguir para realizar algo. Los patrones surgen de la experiencia de seres humanos de tratar de lograr ciertos objetivos. Los patrones capturan la experiencia existente y probada para promover buenas prácticas.

Los patrones expresan esquemas de organización estructural fundamentales para los sistemas de software, estos se ocupan de cuestiones que están más cerca del diseño, la práctica, la implementación, el proceso, el refinamiento, el código. Los patrones arquitectónicos, se han materializado con referencia a lenguajes y paradigmas también específicos de desarrollo.(Craig Larman)

1.2 Arquitecturas más comunes

Generalmente, no es necesario inventar una nueva arquitectura de software para cada sistema de información. Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso en concreto. Así,las arquitecturas mas utilizadas son:

- **Cliente-Servidor**
- **Arquitectura en capas**
- **Orientada a Servicios (SOA por sus siglas en inglés)**

1.2.1 Cliente-servidor

La tecnología Cliente/Servidor es el procesamiento cooperativo de la información por medio de un conjunto de procesadores, en el cual múltiples clientes, distribuidos geográficamente, solicitan requerimientos a uno o más servidores centrales. Desde el punto de vista funcional, se puede definir esta tecnología como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información de forma transparente aún en entornos multiplataforma. Se trata de la arquitectura más extendida en larealización de Sistemas Distribuidos.

En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema. La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni tiene que ser un sólo programa. Una disposición muy común son los sistemas multicapa

en los que el servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras aumentando así el grado de distribución del sistema.

Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores terminales, y los servidores del correo. Mientras que sus propósitos varían en algunos aspectos, la arquitectura básica sigue siendo igual.

Una disposición muy común son los sistemas multicapa en los que el servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras aumentando así el grado de distribución del sistema.

Un sistema Cliente/Servidor es un sistema de información distribuido basado en las siguientes características:

- **Servicio:** unidad básica de diseño. El servidor los proporciona y el cliente los utiliza.
- **Recursos compartidos:** Muchos clientes utilizan los mismos servidores y, a través de ellos, comparten tanto recursos lógicos como físicos.
- **Protocolos asimétricos:** Los clientes inician "conversaciones". Los servidores esperan su establecimiento pasivamente.
- **Transparencia de localización física de los servidores y clientes:** El cliente no tiene por qué saber dónde se encuentra situado el recurso que desea utilizar.
- **Sistemas débilmente acoplados**:Interacción basada en envío de mensajes.
- **Encapsulamiento de servicios:** Los detalles de la implementación de un servicio son transparentes al cliente.
- **Escalabilidad horizontal** (añadir clientes) **y vertical** (ampliar potencia de los servidores). Se puede aumentar la capacidad de clientes y servidores por separado.

- **Integridad:** Datos y programas centralizados en servidores facilitan su integridad y mantenimiento.
- **Centralización del control:** Los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema.

En el modelo usual Cliente/Servidor, un servidor, se activa y espera las solicitudes de los clientes. Habitualmente, múltiples clientes comparten los servicios de un programa servidor común. Tanto los programas clientes como los servidores son con frecuencia parte de un programa o aplicación mayores.

Componentes de la arquitectura Cliente/Servidor

Como se ha venido diciendo, cliente/servidor es un modelo basado en la idea del servicio, en el que el cliente es un proceso consumidor de servicios y el servidor es un proceso proveedor de servicios. Además esta relación está establecida en función del intercambio de mensajes que es el único elemento de acoplamiento entre ambos. De estas líneas se desprenden los tres elementos fundamentales sobre los cuales se desarrollan e implantan los sistemas cliente/servidor: el proceso cliente que es quien inicia el diálogo, el proceso servidor que pasivamente espera a que lleguen peticiones de servicio y el middleware que corresponde a la interfaz que provee la conectividad entre el cliente y el servidor para poder intercambiar mensajes.

Para entender en forma más ordenada y clara los conceptos y elementos involucrados en esta tecnología se puede aplicar una descomposición o arquitectura de niveles. Esta descomposición principalmente consiste en separar los elementos estructurales de esta tecnología en función de aspectos más funcionales de la misma:(R Niella 2009)

- **Nivel de Presentación:** Agrupa a todos los elementos asociados al componente Cliente.
- **Nivel de Aplicación:** Agrupa a todos los elementos asociados al componente Servidor.
- **Nivel de comunicación:** Agrupa a todos los elementos que hacen posible la comunicación entre los componentes Cliente y servidor.

- **Nivel de base de datos:** Agrupa a todas las actividades asociadas al acceso de los datos.

Elementos principales

- **Cliente**

  Un cliente es todo proceso que reclama servicios de otro, el cliente es el proceso que permite al usuario formular los requerimientos y pasarlos al servidor, se loconoce con el término front-end. Este normalmente maneja todas las funciones relacionadas con la manipulación y despliegue de datos, por lo que están desarrollados sobre plataformas que permiten construir interfaces gráficas de usuario (GUI), además de acceder a los servicios distribuidos en cualquier parte de la red.

  Las funciones que lleva a cabo el proceso cliente se resumen en los siguientes puntos:

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

- **Servidor**

  Es el proceso encargado de atender a múltiples clientes que hacen peticiones de algún recurso administrado por él. Al proceso servidor se lo conoce con el término back-end. El servidor normalmente maneja todas las funciones relacionadas con la mayoría de las reglas del negocio y los recursos de datos.

  Las funciones que lleva a cabo el proceso servidor se resumen en los siguientes puntos:

  - Aceptar los requerimientos de bases de datos que hacen los clientes.
  - Procesar requerimientos de bases de datos.
  - Formatear datos para trasmitirlos a los clientes.

- Procesar la lógica de la aplicación y realizar validaciones a nivel de bases de datos.

- **Middleware**

  El middleware es un módulo intermedio que actúa como conductor entre sistemas permitiendo a cualquier usuario de sistemas de información comunicarse con varias fuentes de información que se encuentran conectadas por una red. En el caso que nos concierne, es el intermediario entre el cliente y el servidor y se ejecuta en ambas partes.

  La utilización del middleware permite desarrollar aplicaciones en arquitectura Cliente/Servidor independizando los servidores y clientes, facilitando la interrelación entre ellos y evitando dependencias de tecnologías propietarias. (temariotic.wikidot.com 2009)

Tipos de arquitectura Cliente/Servidor

Uno de los aspectos claves para entender la tecnología Cliente/Servidor, y por tanto contar con la capacidad de proponer y llevar a cabo soluciones de este tipo, es llegar a conocer la arquitectura de este modelo y los conceptos o ideas asociados al mismo. Más allá de entender los componentes cliente/middleware/servidor, es preciso analizar ciertas relaciones entre éstos, que pueden definir el tipo de solución que se ajusta de mejor forma a las estadísticas y restricciones acerca de los eventos y requerimientos de información que se obtuvieron en la etapa de análisis de un determinado proyecto.

Modelos Cliente/Servidor

Una de las más comunes y discutidas distinciones entre las diferentes arquitecturas cliente/servidor se basan en la idea de planos (tier), la cual es una variación sobre la división o clasificación por tamaño de componentes (clientes grandes y servidores amplios). Esto se debe a que se trata de definir el modo en que las prestaciones funcionales de la aplicación serán asignadas, y en que proporción, tanto al cliente como al servidor. Dichas prestaciones se deben agrupar entre los tres componentes clásicos para cliente/servidor;

interfaz de usuario, **lógica de negocios** y **los datos compartidos**, cada uno de los cuales corresponde a un plano.

- **A nivel de software**

  Este enfoque o clasificación es el más generalizado y el que más se ajusta a los enfoques modernos, dado que se fundamenta en los componentes lógicos de la estructura Cliente/Servidor y en la madurez y popularidad de la computación distribuida. Por ejemplo, esto permite hablar de servidores de aplicación distribuidos a lo largo de una red, y no tiene mucho sentido identificar a un equipo de hardware como servidor, si no más bien entenderlo como una plataforma física sobre la cual pueden operar uno o más servidores de aplicaciones.

  - **Modelo Cliente/Servidor 2 capas**

    Las aplicaciones cliente-servidor clásicas o de 2 capas como su nombre lo indica agrupan la lógica de presentación (interfaz) y la lógica de aplicación en la máquina cliente y acceden a fuentes de datos compartidos a través de una conexión de red que se encuentra en el servidor de datos. Estas aplicaciones de 2 capas trabajan bien en aplicaciones a escala de departamentos con un modesto número de usuarios, una base de datos sencilla y una red segura y rápida.

    La ventaja que presenta este tipo de aplicaciones es que los datos están centralizados,lo cual beneficia a la empresa pues es más fácil compartir los datos, se simplifica la generación de reportes y se proporciona consistencia en el acceso a los datos. (temariotic.wikidot.com 2009)

  - **Modelo Cliente/Servidor 3 capas**

    Esta estructura se caracteriza por elaborar la aplicación en base a dos capas principales de software, más la capa correspondiente al servidor de base de datos. Al igual que en la arquitectura dos capas, y según las decisiones de diseño que se tomen, se puede

balancear la carga de trabajo entre el proceso cliente y el nuevo proceso correspondiente al servidor de aplicación.

En este esquema el cliente envía mensajes directamente al servidor de aplicación el cual debe administrar y responder todas las solicitudes. Es el servidor, dependiendo del tipo de solicitud, quien accede y se conecta con la base de datos.

Ventajas:

- Reduce el tráfico de información en la red por lo que mejora el rendimiento de los sistemas (especialmente respecto a la estructura en dos planos).
- Brinda una mayor flexibilidad de desarrollo y de elección de plataformas sobre la cual montar las aplicaciones. Provee escalabilidad horizontal y vertical.
- Se mantiene la independencia entre el código de la aplicación(reglas y conocimiento del negocio) y los datos, mejorando la portabilidad de las aplicaciones.
- Los lenguajes sobre los cuales se desarrollan las aplicaciones son estándares lo que hace más exportables las aplicaciones entre plataformas.
- Dado que mejora el rendimiento al optimizar el flujo de información entre componentes, permite construir sistemas críticos de alta fiabilidad.
- El mismo hecho de localizar las reglas del negocio en su propio ambiente, en vez de distribuirlos en la capa de interfaz de usuario, permite reducir el impacto de hacer mantenimiento, cambios urgentes de última hora o mejoras al sistema.

Inconvenientes:

- Dependiendo de la elección de los lenguajes de desarrollo, puede presentar mayor complejidad en comparación con Cliente/Servidor dos planos.

- Existen pocos proveedores de herramientas integradas de desarrollo con relación al modelo Cliente/Servidor dos planos, y normalmente son de alto costo.

1.2.2 Arquitectura en capas

En este estilo arquitectónico cada capa proporciona servicios a la capa superior y se sirve de las prestaciones que le brinda la inferior, al dividir un sistema en capas, cada capa puede tratarse de forma independiente, sin tener que conocer los detalles de las demás. La división de un sistema en capas facilita el diseño modular, en la que cada capa encapsula un aspecto concreto del sistema y permite además la construcción de sistemas débilmente acoplados, lo que significa que si se minimiza las dependencias entre capas, resulta más fácil sustituir la implementación de una capa sin afectar al resto del sistema. Los sistemas o arquitecturas en capas constituyen uno de los estilos que aparecen con mayor frecuencia mencionados como categorías mayores del catálogo, o por el contrario, como una de las posibles encarnaciones de algún estilo más envolvente.

Garlan y Shaw 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. Instrumentan así una vieja idea de organización estratigráfica que se remonta a las concepciones formuladas por Dijkstra en la década de 1960, largamente explotada en los años subsiguientes. En algunos ejemplares, las capas internas están ocultas a todas las demás, menos para las capas externas adyacentes, y excepto para funciones puntuales de exportación; en la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas.

La programación por capas es un estilo de programación en la que el objetivo primordial es dividir, fraccionar y llegar a separar la Presentación (donde se muestra y se obtienen datos), de la Lógica de Negocio (donde se realizan operaciones) y con todo esto se obtendría independencia, por lo que si hay cambios de arquitectura se puede acceder a los datos o cambiar el negocio o modificar la presentación y solo sería en esa fracción de la capa.

- **Capa de Presentación**

  Es la que ve el usuario, presenta el sistema al mismo, 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. Su función es la de proveer una interfaces para realizar la transferencia de datos.

  Los componentes de la interfaz de usuario deben mostrar datos al usuario, obtener y validar los datos procedentes del mismo e interpretar las acciones de éste que indican que desea realizar una operación con los datos. Asimismo, la interfaz debe filtrar las acciones disponibles con el fin de permitir al usuario realizar sólo aquellas operaciones que le sean necesarias en un momento determinado.

  La capa de presentación se implementa mediante tecnologías JSP, Servlets, con las que se construyen y conectan las interfaces gráficas con la capa de lógica del negocio. Es la encargada de interactuar con el usuario, obtener, validar y enviar los datos al servidor.

  Las interfaces web deben ser consistentes con la información que se requiere, no se deben utilizar más campos de los necesarios, así como la información requerida tiene que ser especificada de manera clara y concisa, no debe haber más que lo necesario en cada formulario. El objetivo principal de este componente es facilitar al usuario la interacción con la misma.

  Dentro de la parte técnica, la capa de presentación contiene los objetos encargados de comunicar al usuario con el sistema mediante el intercambio de información, capturando y desplegando los datos necesarios para realizar alguna tarea.

- **Capa de negocio**

  Es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían 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 recibe solicitudes de la capa de presentación, y presenta los resultados, solicitando a la capa de datos, mediante el gestor de base de datos para almacenar o recuperar datos de él.

Los objetos de negocio son los objetos del Modelo que implementan la lógica de negocio. Sus funciones fundamentales son:

- Realizar la validación de los datos introducidos por el usuario.
- Ejecutar la petición realizada por el usuario. Para ello podrá valerse de transacciones contra Sistemas Host, consultas a bases de datos, consultas a proveedores de contenidos, etc.
- Generar los objetos que utilizarán las Vistas para mostrar los resultados obtenidos.

Su función es garantizar que toda la lógica de negocio esté bien localizada y no mezclada con objetos de otras capas, esto proporciona grandes ventajas, pues define los objetos de negocio, y crea las interfaces de servicio con sus implementaciones.

En esta capa se reciben solicitudes de la capa de presentación, la cual procesa y obtiene los resultados invocando a la capa de Acceso a Datos. La comunicación con otros sistemas que actúan en conjunto, se hace mediante esta capa.

La responsabilidad de esta capa es implementar la lógica de negocio que será consumida por la interfaz del sistema y por otros sistemas a través de los servicios compartidos. Recibe los datos que ingresó el usuario del sistema mediante la capa de presentación, luego los procesa y crea objetos según lo que se necesite hacer con los datos.

- **Capa de acceso a datos**

  Es el puente entre la capa de Lógica de Negocio y el Sistema de Base de Datos. Encapsula la lógica de acceso a datos. Aquí se encuentran componentes que hacen transparente el acceso a la base de

datos. Este es el lugar idóneo para implementar los objetos de acceso a datos (DAOs) -estos encapsulan la persistencia de los objetos de dominios, proveen la persistencia de los objetos transitorios y las actualizaciones de los objetos existentes en la base de datos-, permitiendo ingresar, obtener, actualizar y eliminar información del Sistema de Bases de Datos.

Es donde residen las clases que se encargan de gestionar todo el acceso a la información contenida en los gestores de bases de 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.

Esta capa de acceso a datos se relaciona directamente con el patrón DataMapper pues permite manejar toda la carga y almacenamiento entre el modelo de dominio y la base de datos, logrando que ambos varíen independientemente. DataMapper permite disponer de dos objetos. El primero representa la lógica de negocio. El segundo el acceso a datos. Se trata de realizar una correlación entre objetos de lógica de negocio y tablas de la base de datos.

Las capas de negocio y de datos pueden residir en el mismo ordenador, y si el crecimiento de las necesidades lo aconseja se pueden separar en dos o más ordenadores. Además si el tamaño o complejidad de la base de datos aumenta, se puede separar en varios ordenadores, los cuales recibirán las peticiones del ordenador en que resida la capa de negocio**.**
Si por el contrario fuese la complejidad en la capa de negocio lo que obligase a la separación, esta capa de negocio podría residir en uno o más ordenadores que realizarían solicitudes a una única base de datos. En sistemas muy complejos se llega a tener una serie de ordenadores sobre los cuales corre la capa de acceso a datos.

1.2.3 Orientada a Servicios (SOA)

Es una arquitectura cuyo propósito es permitir que servicios diferentes (lenguaje de programación, tecnología y sistema operativo que lo soporta) sean integrados y vinculados a los procesos de negocio de una organización. (Integración de servicios).(www.microsoft.com 2009)

Elementos para la construcción de un SOA:

- **Operación**: Es la unidad de trabajo o procesamiento en una arquitectura SOA.
- **Servicio**: Es un contenedor de lógica. Estará compuesto por un conjunto de operaciones, las cuales las ofrecerá a sus usuarios.
- **Mensaje**: Para poder ejecutar una determinada operación, es necesario un conjunto de datos de entrada. A su vez, una vez ejecutada la operación, esta devolverá un resultado. Los mensajes son los encargados de encapsular esos datos de entrada y de salida.
- **Proceso de negocio**: Son un conjunto de operaciones ejecutadas en una determinada secuencia (intercambiando mensajes entre ellas) con el objetivo de realizar una determinada tarea.

Puede definirse una aplicación SOA como un conjunto de procesos que contiene los servicios que interactúan entre sí mediante mensajes, para que cada una de las operaciones contenidas en ellos cumpla con un objetivo específico, colaborando a un objetivo común.

La finalidad de la arquitectura SOA es conseguir combinar distintos módulos funcionales para generar aplicaciones de carácter específico, proviniendo todos de servicios preexistentes. Cuanto mayor sea la funcionalidad proporcionada por estos módulos menor será el número de interfaces necesarios para alcanzar el objetivo deseado y cada interfaz conlleva un gasto de procesamiento adicional; sin embargo, cuando los módulos son excesivamente grandes resulta complicada su reutilización. Consecuentemente es necesario alcanzar el nivel de granulación adecuado. La expectativa creada por esta nueva arquitectura es que el coste marginal de creación de la enésima aplicación sea cero dado que el software requerido se

encontrará ya disponible para satisfacer los requisitos; tan sólo se requerirá la coordinación entre los elementos.

Elementos de SOA

- **Funciones**
  - **Transporte**: es el mecanismo utilizado para llevar las demandas de servicio desde un consumidor de servicio hacia un proveedor de servicio, y las respuestas desde el proveedor hacia el consumidor.
  - **Protocolo de comunicación de servicios**: es un mecanismo acordado a través del cual un proveedor de servicios y un consumidor de servicios comunican qué está siendo solicitado y qué está siendo respondido.
  - **Descripción de servicio:** es un esquema acordado para describir qué es el servicio, cómo debe invocarse, y qué datos requiere el servicio para invocarse con éxito.
  - **Servicio:** describe un servicio actual que está disponible para utilizar.
  - **Procesos de Negocio:** es una colección de servicios, invocados en una secuencia particular con un conjunto específico de reglas, para satisfacer un requisito de negocio.
  - **Registro de Servicios**: es un repositorio de descripciones de servicios y datos que pueden utilizar los proveedores de servicios para publicar sus servicios, así como los consumidores de servicios para descubrir o hallar servicios disponibles.

- **Calidad de Servicio**
  - **Política**: es un conjunto de condiciones o reglas bajo las cuales un proveedor de servicio hace el servicio disponible para consumidores.
  - **Seguridad**: es un conjunto de reglas que pueden aplicarse para la identificación, autorización y control de acceso a consumidores de servicios.

- **Transacciones**: es el conjunto de atributos que podrían aplicarse a un grupo de servicios para entregar un resultado consistente.
- **Administración**: es el conjunto de atributos que podrían aplicarse para manejar los servicios proporcionados o consumidos.

Las colaboraciones en SOA siguen el paradigma descubrir, ligar e invocar, donde un consumidor de servicios realiza la localización dinámica de un servicio consultando el registro de servicios para hallar uno que cumpla con un determinado criterio. Si el servicio existe, el registro proporciona al consumidor la interfaz de contrato y la dirección del servicio proveedor.

Entidades presentes en una arquitectura orientada a servicios

- **Roles**
- **Operaciones**
- **Artefactos**

Cada entidad puede tomar uno de los tres roles posibles correspondientes a consumidor, proveedor y/o registro:

- Un consumidor de servicios es una aplicación, un módulo de software u otro servicio que demanda la funcionalidad proporcionada por un servicio, y la ejecuta de acuerdo a un contrato de interfaz.
- Un proveedor de servicios es una entidad accesible a través de la red que acepta y ejecuta consultas de consumidores, y publica sus servicios y su contrato de interfaces en el registro de servicios para que el consumidor de servicios pueda descubrir y acceder al servicio.
- Un registro de servicios es el encargado de hacer posible el descubrimiento de servicios, conteniendo un repositorio de servicios disponibles y permitiendo visualizar las interfaces de los proveedores de servicios a los consumidores interesados.

Las operaciones que pueden llevar a cabo estas entidades son:

- **Publicar**. Para poder acceder a un servicio se debe publicar su descripción para que un consumidor pueda descubrirlo e invocarlo.
- **Descubrir**. Un consumidor de servicios localiza un servicio que cumpla con un cierto criterio consultando el registro de servicios.
- **Ligar e Invocar**. Una vez obtenida la descripción de un servicio por parte de un consumidor, éste lo invoca haciendo uso de la información presente en la descripción del servicio.

1.3 Estilos arquitectónicos

Un estilo arquitectónico es una familia de sistemas de software en términos de un patrón de organización estructural, que define un vocabulario de componentes y tipos de conectores y un conjunto de restricciones de cómo pueden ser combinadas. Para muchos estilos puede existir uno o más modelos semánticos que especifiquen cómo determinar las propiedades generales del sistema partiendo de las propiedades de sus partes.

Los estilos expresan la arquitectura en el sentido más formal y teórico, describen entonces una clase de arquitectura, o piezas identificables de las arquitecturas empíricamente dadas. Esas piezas se encuentran repetidamente en la práctica, trasuntando la existencia de decisiones estructurales coherentes. Una vez que se han identificado los estilos, es lógico y natural pensar en reutilizarlos en situaciones semejantes que se presenten en el futuro. Igual que los patrones de arquitectura y diseño, todos los estilos tienen un nombre: **estilos centrados en los datos**, **estilos de llamada y retorno**, **tubería-filtros**,etc.(Carlos Reynoso 2004)
A continuación se mencionan algunos de los estilos arquitectónicos más utilizados en la actualidad.

1.3.1 Estilos de flujo de datos

Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. Ejemplares de la misma serían las arquitecturas de tubería-filtros y las de proceso secuencial en lote.

- **Tubería y filtros**: una tubería es una popular arquitectura que conecta componentes computacionales a través de conectores, de modo que el procesamiento de datos se ejecuta como un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. Este estilo se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada.
- **Secuencial por lotes:** es la ejecución de un programa sin el control o supervisión directa del usuario (que se denomina procesamiento interactivo). Este tipo de programas se caracterizan por que su ejecución no precisa ningún tipo de interacción con el usuario.

1.3.2 Estilos centrados en datos

Esta familia de estilos enfatiza la integrabilidad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Los sub-estilos característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra.

El estilo centrado en datos resulta apropiado para sistemas que se centran en el acceso y actualización de datos en estructuras de almacenamiento que son compartidos por un número indefinido de componentes consumidores. Una de las propiedades más destacable de este estilo arquitectónico es la necesidad de crear persistencia de los datos almacenados. Existen dos tipos de componentes, un componente que realiza el almacenaje y persistencia de los datos y otros componentes que interactúan con dichos datos.

En estos sistemas hay que tener encuentra que se pueden producir problemas en la interacción de los componentes que manejan los datos con el componente que almacena los datos (creación, lectura y actualización simultáneas a un mismo dato) y la coherencia (hay que evitar que un componente pueda tener un dato con un valor desactualizado). Pueden existir módulos de almacenamiento activos o pasivos según quien se encargue de la notificación de la actualización de los datos (si el componente encargado del

almacenamiento es el notificador, entonces este es un modelo activo). Patrones característicos de este estilo son la pizarra y el repositorio. (Carlos Reynoso 2004)

- **Arquitectura de Pizarra o Repositorio**
  En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él [SG96]. En base a esta distinción se han definidos dos subcategorías principales del estilo:
  1. Si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, el repositorio puede ser una base de datos tradicional (implícitamente no cliente-servidor).
  2. Si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control.

1.3.3 Estilos de llamada y retorno

Este estilo es interesante en los sistemas que se centran en la interacción de un usuario con el propio sistema. Podemos dividir la arquitectura en dos partes, la primera representa la interfaz del usuario con el que este realiza la llamada al sistema, la segunda contiene la lógica de negocio que se realiza tras la correspondiente llamada del usuario.

Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.(Carlos Reynoso 2004)

- **Modelo-Vista-Controlador (MVC)**
- **Arquitecturas en Capas**
- **Arquitecturas Orientadas a Objetos**
- **Arquitecturas Basadas en Componentes**

1.3.4 Estilo basados en eventos

Las arquitecturas basadas en eventos se vinculan históricamente con sistemas basados en actores, demonios y redes de conmutación de paquetes (publicación-suscripción). Los conectores de estos sistemas incluyen procedimientos de llamada tradicionales y vínculos entre anuncios de eventos e invocación de procedimientos. La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa (como se haría en un estilo orientado a objetos) un componente puede anunciar mediante difusión uno o más eventos.

Una característica que tiene que cumplir los elementos incluidos en la arquitectura es que deben ser independientes entre sí, para poder lograr la interacción deseada entre los mismos. Estas comunicaciones son dos a dos, por lo que una de las preocupaciones que hay que tener en cuenta a la hora de diseñar una arquitectura de este tipo es lograr la independencia en la interacción entre dos componentes con otra interacción.

- **Publicador/subscriptor**
- **Cliente/servidor**

1.3.5 Estilo 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. (Carlos Reynoso 2004)

- **Arquitecturas Basadas en Eventos**
- **Arquitecturas Orientadas a Servicios**
- **Arquitecturas Basadas en Recursos**

Después de haber visto los estilos más usados en el mundo de la arquitectura de software es tiempo de pasar a analizar los patrones arquitectónicos, los cuales juegan un papel fundamental en la arquitectura de software, ya que definen la estructura de un sistema software, y a su vez se componen de subsistemas con sus responsabilidades, también tienen una serie de normas o directivas para organizar los componentes del mismo sistema, con el objetivo de facilitar el diseño de tal sistema.

1.4 Patrones de arquitectura

Un patrón o modelo de programación, es una solución a un problema que constantemente aparece en el desarrollo programas. Este trata de describir con detalle la solución del problema en forma de plantilla, aunque no siempre de esta forma se encuentra. (C. Mauricio 2007)

En general, un patrón sigue el siguiente esquema:

- **Contexto**: Es una situación de diseño en la que aparece un problema de diseño.
- **Problema**: Es un conjunto de fuerzas que aparecen repetidamente en el contexto.
- **Solución**: Es una configuración que equilibra estas fuerzas. Esta abarca:
    - Estructura con componentes y relaciones.
    - Comportamiento a tiempo de ejecución: aspectos dinámicos de la solución, como la colaboración entre componentes, la comunicación entre ellos, etc.

1.4.1 Patrón Modelo Vista Controlador

Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Su principal finalidad es mejorar la reusabilidad y que las modificaciones en las vistas impacten en menor medida en la lógica de negocio o de datos.

El **modelo** es el responsable de:

- Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.

- Definir las reglas de negocio (la funcionalidad del sistema).
- Llevar un registro de las vistas y controladores del sistema.
- Si estamos ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo.

El **Controlador** es responsable de:

- Recibir los eventos de entrada.
- Contiene reglas de gestión de eventos, estas acciones pueden suponer peticiones al modelo o a las vistas.

Las **Vistas** son responsables de:

- Recibir datos del modelo y lo muestra al usuario.
- Tienen un registro de su controlador asociado.
- Pueden dar el servicio de "Actualización", para que sea invocado por el controlador o por el modelo cuando es un modelo activo.

Este patrón se ve frecuentemente en aplicaciones Web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el controlador es el Sistema de gestión de base de datos y el modelo es el modelo de datos. A lo largo de los años, desde la presentación de este patrón a la comunidad científica se han desarrollado 3 variantes fundamentales, que se presentan brevemente a continuación.

Variante I: (Figura 1).
Variante en la cual se diversifica las funcionalidades del Modelo teniendo en cuenta las características de las aplicaciones multimedias, donde tienen un gran peso las medias utilizadas en estas.

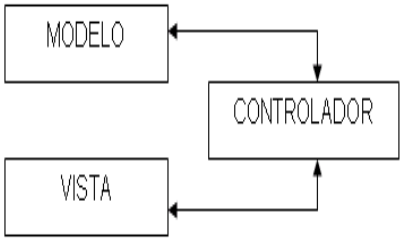


Figura 1. Variante inicial del patrón MVC.

Variante II: (Figura 2).

Variante en la cual se desarrolla una comunicación entre el Modelo y la Vista, donde esta última al mostrar los datos los busca directamente en el Modelo, dada una indicación del Controlador, disminuyendo el conjunto de responsabilidades de este último.

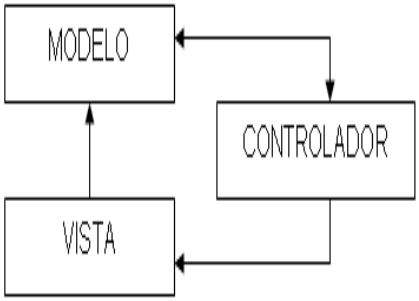


Figura 2. Variante Intermedia del patrón MVC.

Variante III: (Figura 3)

Variante modificada para Aplicaciones Multimedia del MVC.

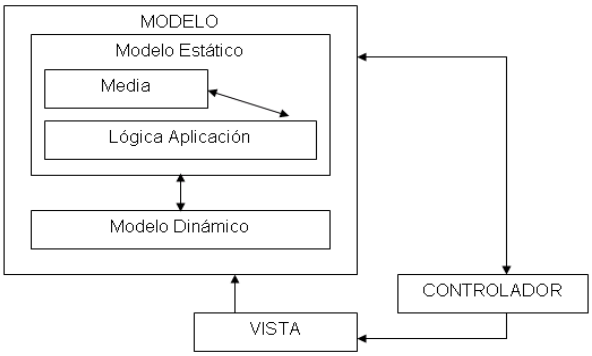


Figura 3. Variante modificada para Aplicaciones Multimedia del MVC.

1.4.2 Patrón capas

La funcionalidad principal de este patrón es crear una modularidad del sistema asignando a cada capa funcionalidades específicas y bien definidas.

- **La capa de la Presentación**

  Esta capa reúne todos los aspectos del software que tiene que ver con las interfaces y la interacción con los diferentes tipos de usuarios. Estos típicamente incluyen el manejo y aspecto de las ventanas, el formato de los reportes, menús, gráficos y elementos multimedia en general.

- **La capa del Dominio de la Aplicación**

  Esta capa reúne todos los aspectos del software que automatizan o apoyan los procesos de negocio que llevan a cabo los usuarios. Estos aspectos típicamente incluyen las tareas que forman parte de los procesos, las reglas y restricciones que aplican. Esta capa también recibe el nombre de la capa de la Lógica de la Aplicación.

- **La capa del Acceso a Datos**

  Esta capa reúne todos los aspectos del software que tienen que ver con el manejo de los datos persistentes, por lo que también se le denomina la capa de las Bases de Datos.

Existen especificaciones de este patrón donde se manipula como será la comunicación entre cada una de las capas definidas.

1.5 Características de la herramienta gestora de proyectos dotProject

El dotProject es una herramienta de gestión de proyectos creada por la comunidad de Software Libre Internacional, es una aplicación basada en la Web, un sistema multiusuario e internacionalizado en varios idiomas, fue construida con aplicaciones de código abierto y puede integrarse a LDAP, lo cual ha facilitado que el administrador de la aplicación no tenga que agregar manualmente a los usuarios, pues cuando este último se loguea automáticamente se adiciona a la base de datos de la aplicación. Además está programado en PHP y utiliza inicialmente MySQL como base de datos (aunque otros motores como PostgreSQL también pueden ser utilizados).

La plataforma recomendada para utilizar dotProject se denomina LAMP (Linux + Apache+ MySQL+ PHP), esto no significa que no se pueda instalar en Windows, pues esta herramienta es multiplataforma.

Hay que tener en cuenta que el sistema de tickets por ejemplo manda correos con información y mensajes, así que es necesario algún servidor de correo corriendo en el servidor si se desea tener habilitada esta opción. El dotProject posee todas las opciones básicas de cualquier gestor de proyectos:

- **Gestor de empresas**
- **Proyectos**
- **Tareas**
- **Calendario**
- **Ficheros**
- **Contactos**
- **Usuarios**
- **Sistema**

Esta herramienta permite gestionar las distintas fases y tareas que componen un proyecto. Además se perfila como una interesante herramienta para trabajar en entornos colaborativos, permitiendo a los integrantes del equipo trabajar compartiendo información relativa a los proyectos. La última versión "estable" de dotProject, es la 2.1.4 y fue liberada en Junio del 2007. Existen dos tipos de distribuciones que dependen de la plataforma sobre la que se pretende instalar el producto (Linux o Windows).

Características Principales:

- Permite la gestión y planificación de proyectos en entornos colaborativos.
- Basado en plataforma Web permite la participación online de los miembros de un proyecto.
- Permite la asignación de recursos (equipamientos, mobiliario...) a un proyecto o varios, así como la descomposición en tareas.
- Permite Vista de eventos y tareas en calendario.
- Permite la visualización de informes y estadísticas sobre los proyectos registrados.
- Gestiona las actividades de la empresa al permitir la parcelación del proyecto en tareas.
- Permite clasificar y/u ordenar los proyectos en función de su estado: En curso,pendientes, cerrados.
- Permite la visualización de informes y estadísticas sobre los proyectos registrados, ejemplo: Las horas asignadas (por usuario o proyecto), estado de un proyecto, estadísticas, etcétera.

Requerimientos de hardware

La herramienta dotProject puede ser instalada en cualquier servidor en el que se encuentren corriendo un servidor de base de datos MySQL o PostgreSQL, y un servidor Web Apache, pero esto depende de la cantidad de usuarios que se conecten.(*ver figura 4*)

Pero para el eficiente funcionamiento de dotProject se requiere de un servidor que posea un 1GB o más de memoria RAM y 2.8 Ghz a fin de contar con tiempos de respuesta exitosos; propuesta validada con 50 usuarios conectados y trabajando simultáneamente.

En la figura 4 se muestra las relaciones físicas entre los componentes de hardware y software que conforman el sistema. Pero a modo de aclaración, el "Servidor de BD" puede estar en el mismo servidor web, pero esto no es aconsejable, pues si tenemos varias base de datos en el mismo servidor, el rendimiento de las aplicaciones web que se encuentren alojadas en ese servidor decrementaría. Además desde el punto de vista de seguridad, lo recomendable es tener el servidor de base de datos separado del servidor de aplicaciones, pues existen procesos de autenticación y autorización en el servidor de aplicaciones que se agregan a los procesos ya existentes en los otros servidores, - base de datos y servidor Web- con lo que el hackear el servidor de aplicaciones, no da acceso al servidor de base de datos.

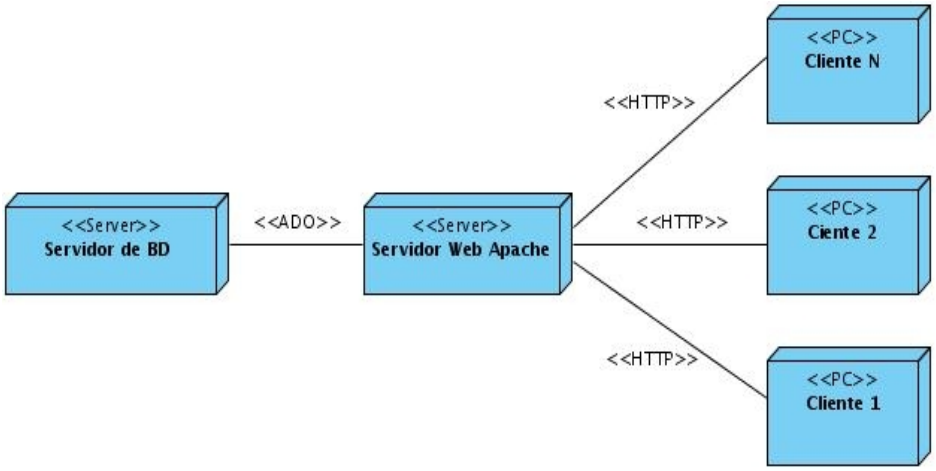


Figura 4. Diagrama de despliegue del dotProject

1.6 Herramientas, lenguajes y tecnologías a utilizar

Tener completa una arquitectura de software implica obtener un producto de calidad. Unos de los aspectos más importantes en la arquitectura es definir las herramientas a usar para el proyecto y el lenguaje de modelado a emplear. Para el desarrollo de este trabajo se utilizaron las siguientes herramientas:

- **Herramienta CASE: Visual Paradigm**
- **Editor de Diagramas: dia**

- **Diseñador de Base de Datos: DBDesigner 4**
- **Lenguaje de programación web:PHP en su versión 5.0**
- **Herramienta de desarrollo web: Quanta+**

1.6.1 Lenguaje de Modelado utilizado.

La arquitectura, conformada por diferentes visiones del sistema,constituye un modelo de como está estructurado dicho sistema, sirviendo de comunicación entre las personas involucradas en el desarrollo y ayudando a realizar diversos análisis que orienten el proceso de toma de decisiones. Para que la arquitectura se convierta en una herramienta útil dentro del desarrollo y mantenimiento de los sistemas de software es necesario que s e cuente con una manera precisa de representarla.

El lenguaje de modelado escogido para utilizar enla representación de la arquitectura fue el Lenguaje de Modelado Unificado (UML), el cual es un lenguaje gráfico para visualizar, especificar, construir y documentar cada una de las partes que comprende el desarrollo de software. UML posee formas de modelar conceptos como lo son procesos de negocio y funciones de sistema, además de aspectos concretos como lo son escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reusables. (Jesus Villamor 2009)

1.6.2 Herramienta CASE: Visual Paradigm

Visual Paradigm utiliza el UML como lenguaje de modelado. Está disponible en varias ediciones: Enterprise, Profesional, Community, Standard, Modeler y Personal, cada una destinada a necesidades específicas. Es una herramienta de gran alcance, fácil de utilizar y que apoya el ciclo de vida completo del software- análisis, diseño, codificación, prueba y despliegue-, además permite dibujar diagramas UML, generar código de diagramas de la clase y viceversa, y generar la documentación soporta todos los diagramas de la más reciente notación de UML.

Fue seleccionada la edición Community de Visual Paradigm que es gratuita, y soporta la versión 2.0 de UML. Esta edición soporta el ciclo de vidacompleto del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Además ayuda a una más rápida construcción de aplicaciones de calidad, y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.(www.visual-paradigm.com 2009)

1.6.3 Editor de Diagramas: dia

Dia es una aplicación para la creación de diagramas técnicos. Entre sus características se incluye: soporte para distintos tipos de diagramas como diagramas entidad-relación, diagramas de red, diagramas de flujo, diagramas de clases, entre otros. Permite la utilización de formas personalizadas creadas por el usuario como simples descripciones XML. Y además, brinda la posibilidad de exportar a múltiples formatos (EPS, SVG, CGM, PNG, etc.).

1.6.4 Diseñador de Base de Datos: DBDesigner 4

DBDesigner 4 es un completo sistema visual para el desarrollo de aplicaciones de bases de datos que integra diseño, modelado, creación y mantenimiento en un ambiente simple y fácil de usar. Además esta disponible para Windows y Linux, DBDesigner es una de las mejores herramientas para la gestión, creación y diseño de bases de datos, entre las funciones que posee destacan: edición en modo diseño, ingeniería inversa para bases de datos MySQL, Oracle, MSSQL y cualquier otra base ODBC, generación de esquema definido por el usuario, soporte para índices, inclusión automática de llaves foránea, capacidad completa de documentación, uso de tipos de datos definidos por el usuario y docenas más de otras opciones, que hacen que nos centremos más en la funcionalidad de nuestra base de datos que en los detalles técnicos.

1.6.5 Lenguaje de programación web: PHP 5

El **PHP** es un lenguaje de programación interpretado, diseñado originalmente por Rasmus Lerdof en 1994, para la creación de páginas web dinámicas. Generalmente se ejecuta en un servidor web, tomando el código en PHP como su entrada y creando páginas web como salida. Puede ser desplegado en la mayoría de los

servidores web y en casi todos los sistemas operativos y plataformas sin costo alguno. Es usado principalmente en interpretación del lado del servidor pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica usando las bibliotecas QT o GTK+. Además soporta ODBC (el Estándar Abierto de Conexión con Bases de Datos), así que puede conectarse a cualquier base de datos que soporte tal estándar. La más reciente versión principal del PHP fue la versión 5.2.6 del 1 de mayo del 2008.(www.php.net/ 2009)

1.6.6 Herramienta de desarrollo web: Quanta +

El Quanta Plus (*Quanta+*) es una herramienta libre de desarrollo de páginas web diseñado para el proyecto KDE que rápidamente se está convirtiendo en un editor maduro que cuenta con varias funcionalidades. Su versión actual es la 3.5 y proporciona un interfaz de múltiples documentos (MDI) poderoso e intuitivo para los desarrolladores web. Puede incrementar exponencialmente la productividad, a través del uso de acciones personalizadas, guiones y barras de herramientas, con lo que se puede automatizar casi cualquier tarea. (Eric L. Laffoon 2004)

1.7 Conclusiones Parciales

En este capítulo se han analizado algunos aspectos teóricos que serán de gran ayuda para el desarrollo de los próximos capítulos, ya que es de suma importancia tenerlo en cuenta para proponer una solución. Se profundizó en el conocimiento de algunos conceptos necesarios para la compresión de este trabajo. Además se realizó un análisis completo de las tecnologías que serán utilizadas a lo largo del desarrollo del sistema propuesto.

2 Capítulo 2 Descripción de la arquitectura

2.1 Descripción de la arquitectura

En un sistema la arquitectura es lo más importante, ya que permite la comunicación entre las personas involucradas, la documentación temprana de las decisiones de diseño, restringe la implementación, facilita las propiedades del sistema, y permite predecir cualidades del sistema.

La herramienta dotProject posee el estilo de arquitectura en capas, este estilo ha sido el predominante para la construcción de aplicaciones multiplataforma altamente modulables, ya que permite eliminar importantes problemas de escalabilidad, disponibilidad, seguridad e integración, además de que simplifica la comprensión y organización de la herramienta. En este estilo arquitectónico a cada nivel se le confía una misión simple, lo que permite el diseño de una arquitectura escalable, o sea, se puede ampliar con facilidad en caso de que las necesidades del sistema aumenten.

Además permite que el desarrollo se pueda llevar a cabo en varios niveles y en caso de algún problema o cambio se ataca al nivel requerido sin tener que revisar entre el código mezclado. Otra de las ventajas de este estilo arquitectónico es que reduce las dependencias de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores, existiendo independencias entre las capas para cualquier detalle.

La herramienta dotProject presenta tres capas, la cual permite que la interfaz del cliente no sea requerida para comprender o comunicarse con el receptor de los datos. Por lo tanto, esa estructura de los datos puede ser modificada sin cambiar la interfaz del usuario. El código de la capa intermedia puede ser reutilizado por múltiples aplicaciones si esta diseñado en formato modular. (*ver figura 5*)

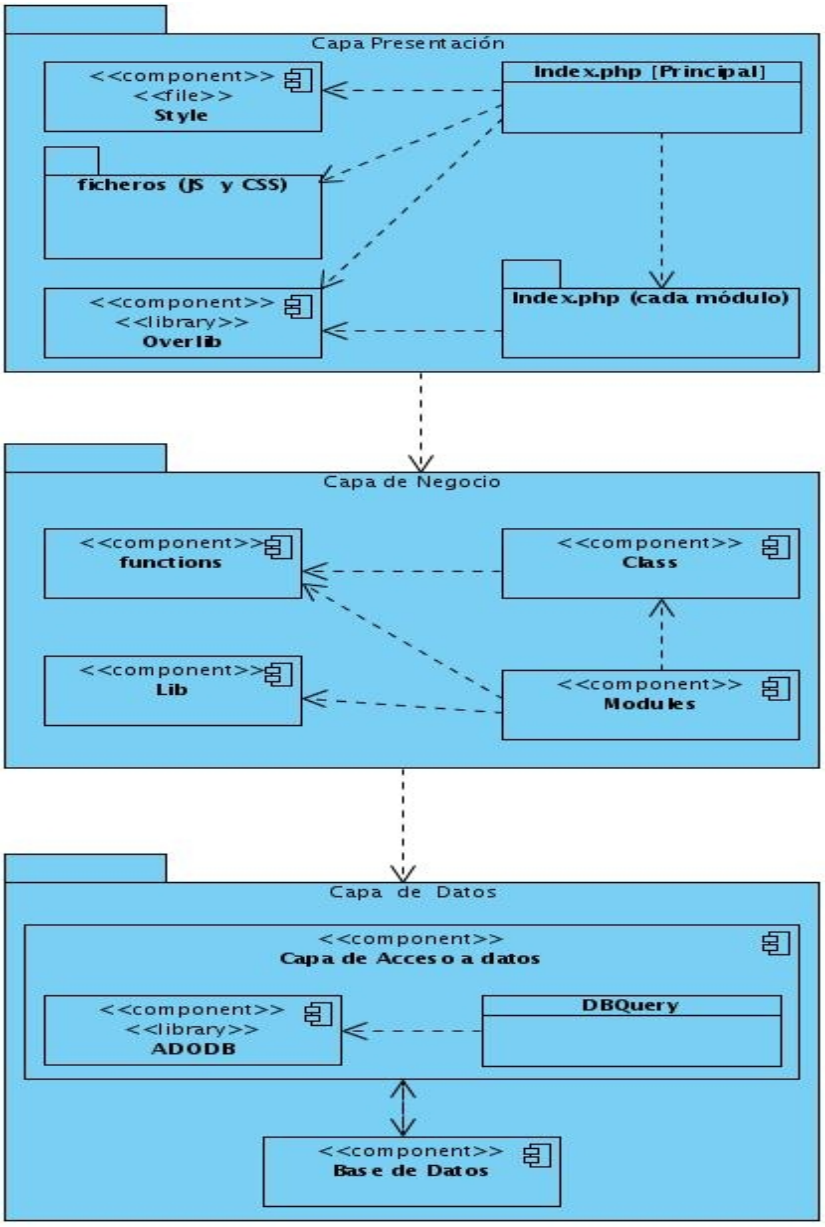


Figura 5. Arquitectura del dotProject

Capítulo 2. Descripción de la arquitectura

A continuación se explicará la función de cada una de las capas,estas son:

- **Capa de presentación o interfaz de usuario.** (*ver figura 6*)

  Es la capa que ve el usuario, muestra una estructura de contenidos a partir de la arquitectura de la información definida para el sistema. La misma contiene todos los ficheros que manipulan elementos de interfaz de usuario como formularios, tablas, capas, elementos Javascript y los estilos CSS correspondientes.

  Esta capa se comunica únicamente con la capa de negocio. También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" para el usuario. En esta capa se encuentran el directorio **style**, los ficheros **index.php** de cada uno de los módulos, el **index.php** del directorio principal, la librería **overlib** del directorio **lib** y todos los ficheros específicos de los módulos que contengan código Javascript y CSS.

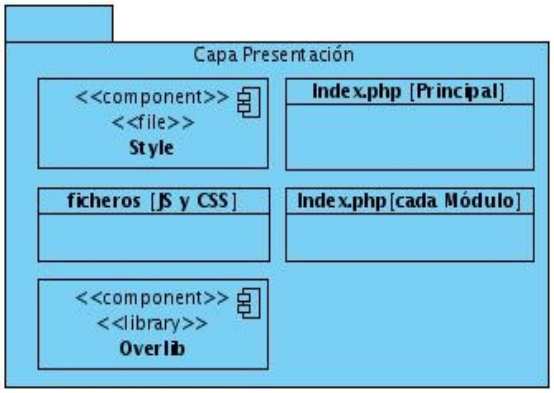


Figura 6. Capa de Presentación

- **Capa de negocio.** (*ver figura 7*)

  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 almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación.

Esta capa está formada por las entidades presentes en el negocio y que representan a los objetos que van a ser manejados o consumidos por toda la aplicación. En este caso, están representados por todas las clases del directorio **class**, todas las librerías existentes en el directorio **lib**, exceptuando a las librerías **overlib y ADODB**, por el directorio **functions** y el directorio **Modules** exceptuando el **index.php** y los **ficheros** que contengan código CSS y JavaScript (en caso de que existan) de cada uno de los módulos.

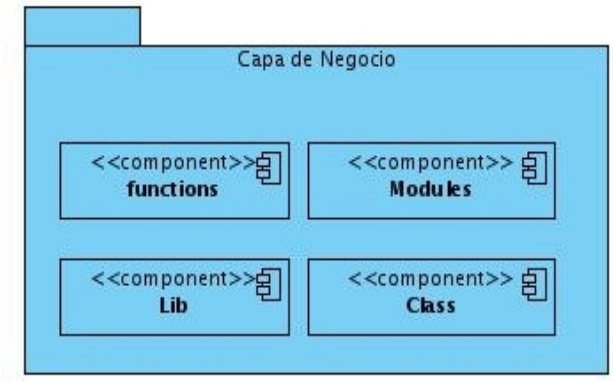


Figura 7. Capa de Negocio

- **Capa de Acceso a Datos** (Ver figura 8)

  Esta capa esta formada por dos subcapas:

  - **Capa de acceso a datos**

    Es la capa que contiene una porción de código que justamente realiza el acceso a los datos. De esta manera cuando es necesario cambiar el motor de base de datos, solamente tendremos que corregir esa capa.

    Esta capa contiene las clases que interactúan con la base de datos, permite realizar todas las operaciones con la base de datos de forma transparente para la capa de negocio. En el caso de dotProject el acceso a la base de datos se realiza mediante la clase DBQuery que constituye una capa de abstracción de la libreria ADODB.

- **Capa de datos**

  En esta capa es donde están los datos y se corresponde directamente con la definición de esquemas, tablas, vistas, procedimientos almacenados y todo lo que se pueda o deba poner en un motor de base de datos. (en el caso de dotProject el motor de base de datos es MySQL).

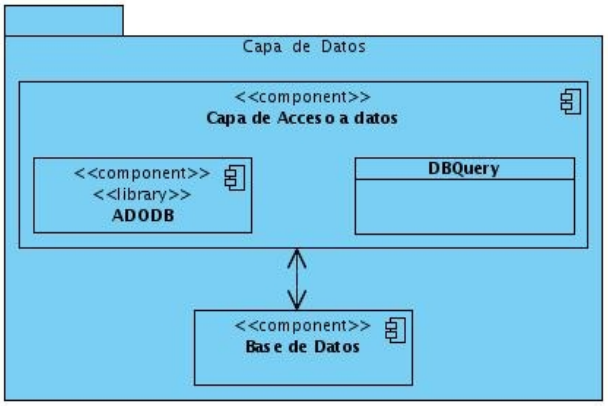


Figura 8. Capa de acceso a datos

2.2 Modelo de Dominio

Para comprender mejor el funcionamiento del dotProject se elaborará un modelo de dominio, el cual contiene los tipos de objetos más importantes del sistema, estos objetos representan la relación que existe entre los componentes fundamentales del sistema. (*ver figura 9*)

2.3 Estructura de directorios del dotProject

Los principales componentes del dotProject se presentan en forma de archivos y directorios.(*ver figura 10*)
Los tres únicos archivos que se encuentran en la carpeta raíz de la instalación son :

- **Archivos de cambios:** que sirven de historial del sistema para el intercambio de sus versiones.
- **fileviewer.php:** es un script para mostrar archivos utilizados por todo el sistema.

- **index.php:** es el principal archivo del entorno que controla los mensajes de error, por ejemplo, los objetos de la clase base, los envíos de las principales variables y la pantalla de configuración, además de buscar el idioma que se utilizará.

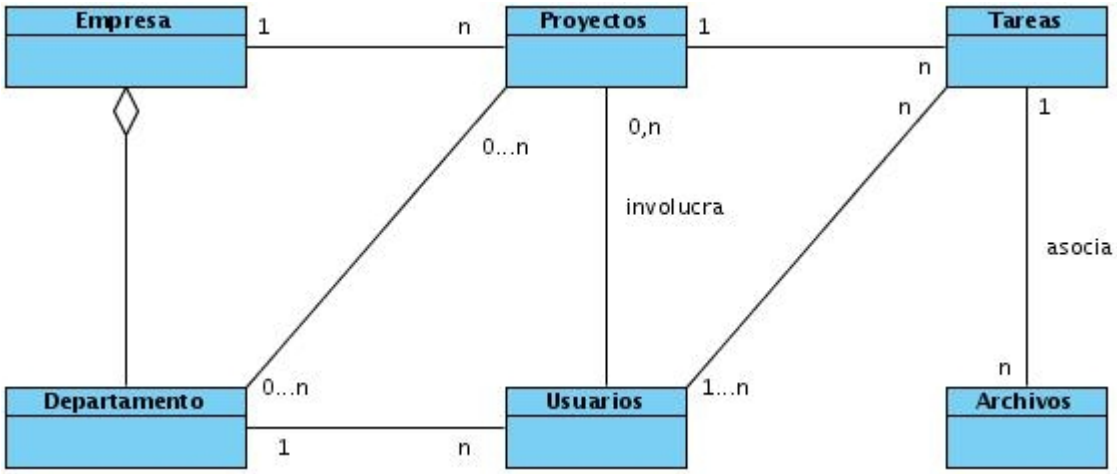


Figura 9. Modelo de dominio del dotProject

Directorio lib.

En este directorio se encuentran un conjunto de librerías que son utilizadas por el dotProject para su correcto funcionamiento, el número de librerías en este directorio depende de las funcionalidades que se desee tener habilitadas en la aplicación.

A continuación se mencionarán algunas de las librerías más comunes en este directorio:

- **Adodb**: Es un conjunto de librerías de bases de datos para **PHP** y **Python**. Esta permite a los programadores desarrollar aplicaciones Web de una manera potable, rápida y fácil. La ventaja reside en que la base de datos puede cambiar sin necesidad de reescribir cada llamada a la base de datos realizada por la aplicación. Esta librería soporta variedades de bases de datos, pero en nuestro caso solo interesan las bases de datos Mysql, PostgreSql y LDAP.

- **Calendar**: Esta librería como bien lo indica su nombre, posee una numerosa cantidad de utilidades para trabajar con el calendario del dotProject.
- **Date:** Está compuesta por un conjunto de clases genéricas que son utilizadas para la representación y la manipulación de fechas, horas y zonas horarias, lo cual es una gran limitación para programas PHP. Además, incluye datos de zona horaria y muchas conversiones de fechas o de horas y permite visualizar y comparar los calendarios de fechas antes de 1970 y posterior a 2038.
- **Ezpdf:** Esta librería nos permite crear de manera dinámica documentos pdf con php, sin necesidad de usar algún tipo de módulo, es usada principalmente para generar los informes del módulo de Reportes.
- **Jpgraph**: es una librería de clases orientadas a objetos para PHP (recomendada para PHP 4.3.0), para la creación dinámica de imágenes. Con Jpgraph se pueden crear gráficas complejas con un mínimo de código y con un control muy detallado.(Mario Alberto Arredondo Guzmán 2009)
- **Pear:** Es un entorno de desarrollo y sistema de distribución para componentes de código PHP que son utilizados en la reutilización de código que realizan tareas comunes tales como la librería **Date**, y la elaboración información de los contactos.
- **Overlib:** Es una librería Javascript creada para mejorar las Web con pequeñas ventanas "popup" informativas que ayudan a los usuarios del dotProject suministrándole información de lo que sucederá cuando pulse en un enlace. (Erik Bosrup 2009)
- **Phpgacl:** Es un proyecto para proporcionar a los desarrolladores Web un sistema simple, pero muy potente, que se utiliza para manejar los permisos de acceso, es decir para tratar los permisos en dependencia de los roles existentes en la aplicación.
- **Smarty**: es un motor de plantillas Open Source para PHP que lleva muchos años en el mercado. Con él podremos realizar aplicaciones web de calidad separando el código (PHP) de la presentación (HTML/CSS). (maestrosdelweb.com 2007)

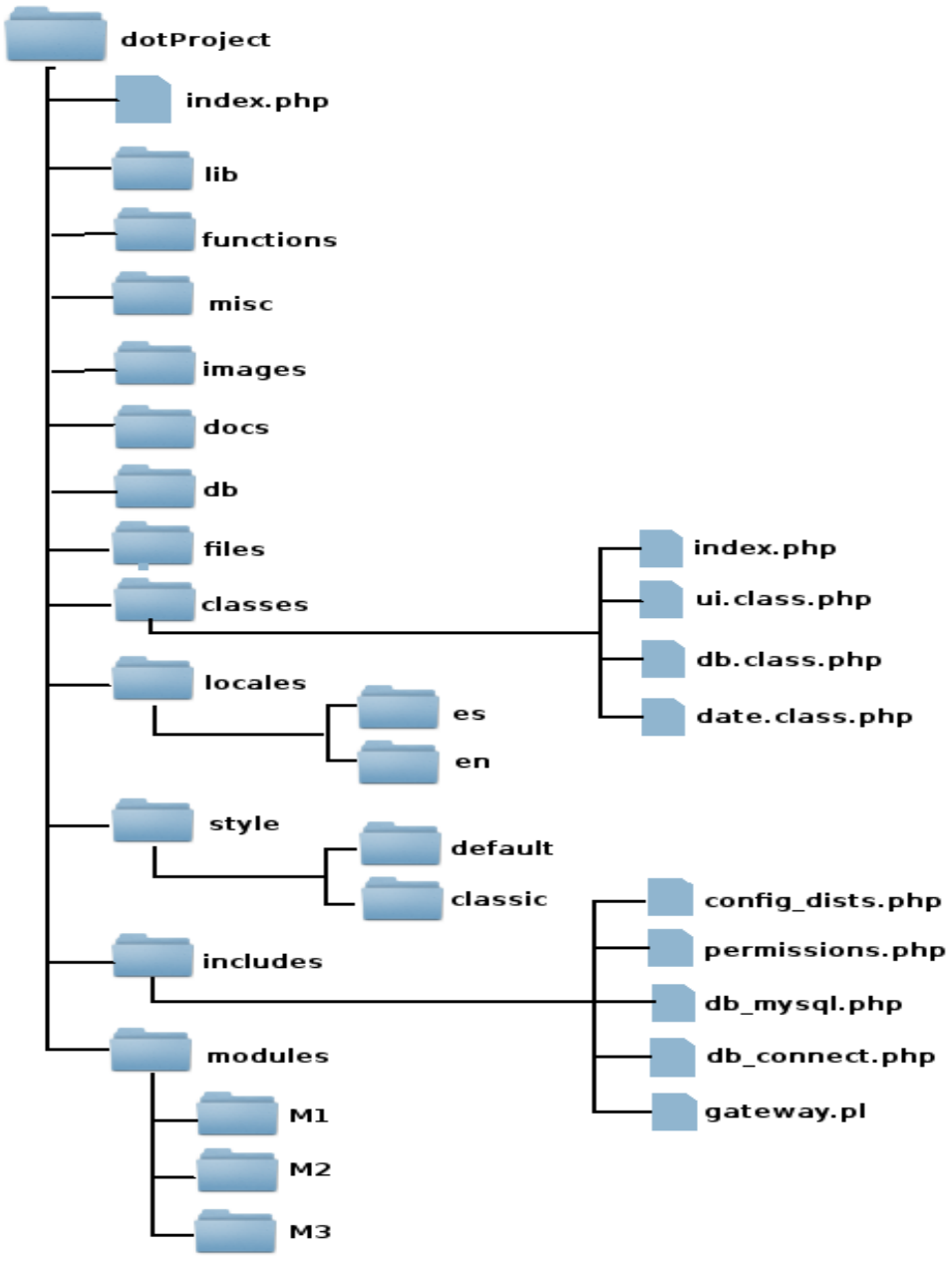


Figura 10. Estructura de directorios del dotProject

Directorio functions

En este directorio es donde se almacenan las funciones básicas que utiliza el dotProject, ejemplo de esas funciones son:

- **admin_func.php:** esta función es la responsable de asignar en la base de datos los permisos de administrador del sitio a los usuarios designados.
- f**orums_func.php**: esta función se encarga de verificar si el usuario esta registrado o no en la base de datos, limitando los permisos de acceso a los foros, es decir, si el usuario no se encuentra registrado solo tendrá permiso de ver los temas visitados en los últimos 30 días, pero si por el contrario el usuario está registrado, este tendrá acceso a los foros,a los vistos, a los proyectos y a la empresa a la cual pertenece y a los proyectos inactivos.
- **projects_func.php:** esta función es la encargada de establecer la correspondencia entre un color y la prioridad de los proyectos.
- **tasks_func.php:**esta función contiene variables que serán utilizadas por el módulo de tareas, entre esas variables se encuentran el porcentaje,el estado y la prioridad de una tarea o de varias.

Directorio misc

En este directorio se encuentra el módulo de autenticación para el dotProject, y se utiliza para almacenar archivos y funciones generales de la herramienta gestora de proyectos.

Directorio images

En este directorio es donde se almacenan todas las imágenes que son utilizadas por el dotProject.

Directorio db

Este directorio se utiliza para almacenar archivos usados para la instalación o actualización del entorno, como los archivos de ayuda y scripts para la base de datos.

Directorio files

En este directorio se almacenan los archivos que se añaden temporalmente al entorno por los usuarios para resaltar una función o tarea realizada.

Directorio classes

En este directorio se encuentran todas las clases que utiliza dotProject, estas clases se pueden clasificar en dos grupos principales: El primero consiste en una serie de clases que constituyen una capa de abstracción de algunas de las librerías que utiliza dotProject. Estas son:

- **Permissions.class.php:** Esta hereda de la clase phpgacl. No contiene muchos cambios pero el principal de ellos consiste en que provee la base de datos para proporcionar detalles del entorno de dotProject.
- **Query.class.php:** Constituye una capa de abstracción de la librería de acceso a datos adodb y facilita un conjunto de funcionalidades que proporcionan al programador mayor versatilidad, comodidad y nivel de abstracción a la hora de utilizar esta clase.
- **Date.class.php:** Esta clase es la encargada de realizar las operaciones con fechas como bien lo indica su nombre y constituye una capa de abstracción del paquete **Date** de la librería **Pear.** Entre las principales funcionalidades que posee están las de obtener la diferencia en días entre dos fechas, conocer si una fecha es mayor que otra y agregar días y/o meses a la fecha actual.

El otro grupo de clases son implementaciones de distintas funcionalidades que requiere el sistema para su correcto desarrollo. Estas son:

- **Authenticator**: Esta clase se encarga de implementar los tres tipos de autentificación que posee dotProject. Estos son PostNuke, directamente contra la BD y por un servidor LDAP, siendo este último el más popular.
- **CustomFields**: Es para la creación de componentes HTML de formas dinámicas.

- **Event_Queue:** Es una cola donde se almacenan los eventos a realizarse, tales como notificaciones de eventos y el tiempo de otros eventos, así como los mensajes de correos electrónicos salientes.
- **LibMail**: Esta clase es utilizada para el envío de correo, esta posee una serie de funciones básicas para realizar el envío como son: Subject(), From(), ReplyTo(), To(), Body(), Attach(), entre otras.
- **UI:** Esta clase es la encargada de gestionar los elementos de interfaz de cada usuario que esté logueado. Manipula información personal del usuario como son el nombre completo, compañía, departamento, email y su identificador de conexión.

Directorio locales

En este directorio es donde se guardan los distintos idiomas con los que cuenta dotProject.

Directorio style

En este directorio se encuentran los diferentes temas disponibles para el dotProject, el tema habilitado depende de la configuración que le predefinan los usuarios.

Directorio includes

Este directorio se utiliza para almacenar todos los archivos importantes y los permisos de los mismos, además de la configuración de la base de datos, las configuraciones del entorno y la secuencia de scripts para correo electrónico. Es decir, en este directorio se guardan los ficheros de configuración que son utilizados por el dotProject, tales como:

- **Db_adodb**: este fichero de forma general se encarga de recoger los datos, para ello consta de una serie de funciones tales como:
  - **db_connect**($host='localhost', $dbname, $user='root', $passwd='', $persist=false) //esta función es la encargada de conectar a la base de datos.

- **db_error** () //esta función es la encargada de devolver en caso que exista algún error en la conexión a la base de datos donde se encuentra el mismo.
- **db_insert_id** () //se encarga de insertar objetos en la base de datos.
- **Sendpass**: Este fichero será el responsable de enviar las nuevas contraseñas de los usuarios a la base de datos y envía un correo con dicha contraseña si todos los parámetros están correctos, en caso de no estarlo retornaría un mensaje de error mostrándonos donde ocurrió este.
- **Session:** En este fichero se encuentran una serie de funciones para el manejo de sesiones, vale aclarar que estas funciones serán válidas siempre y cuando la base de datos sea accesible y exista una tabla llamada "sessions" ejemplo de estas funciones son:
  - **dPsessionOpen()**
  - **dPsessionClose()**
  - **dPsessionRead()**
  - **dpSessionStart()**

- **Permissions**: En esta clase se definen los permisos que tendrá cada usuario, en este fichero se definen tres banderas de permisos que se utilizaran en esta clase, estas son:
  - **define('PERM_DENY', '0')**
  - **define('PERM_EDIT', '-1')**
  - **define('PERM_READ', '1')**

  Por defecto todas las cuentas tienen permiso de editar. Esta clase posee una serie de funciones tales como:
  - **checkFlag ()** // Esta función se utiliza para comprobar los permisos.

- **isAllowed ()** // Esta función verifica determinados permisos de un módulo, y opcionalmente, el id.
- **Db_connect**: Esta clase es la encargada de gestionar la conexión a la base de datos.

- **Config**: Esta clase contiene los archivos de configuración generados automáticamente por la instalación del dotProject.

Directorio Modules

Este directorio es el más importante, ya que mantiene la totalidad de los módulos del dotProject en forma de directorios. Estos directorios tienen el nombre del módulo y dentro de cada directorio tiene los archivos necesarios para ejecutar dicho módulo. (*ver figura 11*)

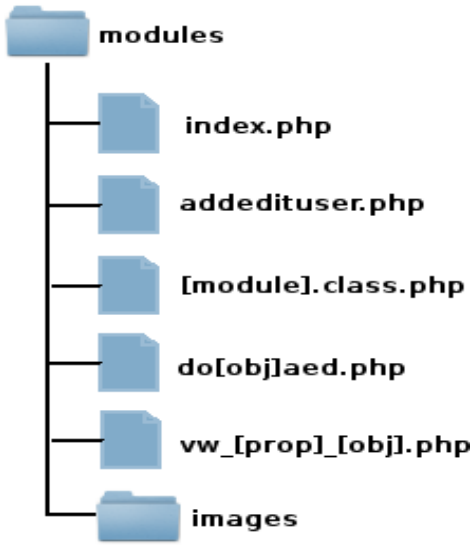


Figura 11. Estructura básica de un módulo de dotProject y los archivos que lo componen.

El archivo**[modules].class.php-,**donde [**modules**] es el nombre del módulo - se utiliza para desarrollar una clase base del módulo.

El archivo **addedit [obj].php** es sólo para la creación de formas de inclusión/exclusión de objetos de dicho módulo, donde cada módulo puede tener varios objetos.

El archivo **do_[obj]_aed.php** realiza los procedimientos de la base de datos para su inclusión, el cambio y la exclusión de los objetos en el módulo.

El módulo también puede contener un directorio llamado **images** para almacenar las imágenes específicas de ese módulo. Además de estos archivos, existen dos más que intervienen en la creación de un módulo, estos son:

- **index.php** : que se utiliza para llenar y utilizar la página principal del módulo.
- **vw_[prop]_[obj].php:** para definir diversas visualizaciones de propiedades u objetos diferentes.

La cantidad de módulos que se encuentra en este directorio varia en dependencia de los módulos que se quieran tener instalados, entre los más comunes están:

- **Companies**: Identificación que se le asigna a las empresas/administraciones que conforman la organización. Los tipos predefinidos que tiene la herramienta son: Cliente, Vendedor, Proveedor, Consultor, Gobierno e Interno. Cada empresa debe tener un dueño asociado, según términos de la herramienta, que realmente identifica a los responsables con mayor orden jerárquico de la empresa. Quizás el nombre de empresa confunda o choque con los términos utilizados en la organización, pueden entenderse también como Gerencias, Direcciones, Coordinaciones, entre otras. Entre las operaciones generales de este módulo se encuentran:
  - Visualizar las empresas.
  - Crear nuevas empresas.
  - Editar una empresa.
  - Eliminar una empresa.
- **Projects:** Es la entidad que contiene el grupo de tareas necesarias para desarrollar un determinado producto. Una empresa puede tener varios proyectos asociados. Este módulo es el que permite crear

nuevos proyectos. Se puede especificar multitud de detalles cuando se crea un nuevo proyecto: nombre, empresa propietaria, fechas, número de tareas, estado, presupuesto inicial, presupuesto actual, departamento encargado de la realización del proyecto. Asimismo se pueden importar las tareas de alguno de los proyectos ya existentes dentro de la organización. Posteriormente se puede visualizar el estado de cada uno de los proyectos, sus diagramas de Gantt, modificar el estado, etc.

- **Tasks**: Este módulo se utiliza por un lado para visualizar las tareas pendientes del usuario de la herramienta y por otro para crear nuevas tareas y asignarlas si se es el jefe del proyecto. Desde el primer punto de vista, aparecen la lista de tareas de todos los proyectos en los que esta asignada el individuo. Por cada una de estas tareas se muestra el porcentaje de completitud, el usuario que creó la tarea, la fecha de inicio, la duración estimada, los recursos materiales asignados, el diagrama Gantt para la tarea, así como la posibilidad de generar un informe sobre la tarea. Desde el punto de vista del gestor del proyecto, se pueden crear tantas tareas como sea necesario, así como subtareas dentro de cada tarea, dependencias entre las tareas, así como ficheros para cada tarea.
- **Calendar**: Es la típica aplicación de calendario en la que se muestran todos los eventos importantes que puedan ocurrir en el día: tareas por realizar, reuniones, eventos, etc.
- **Files**: A través de esta aplicación se gestiona todos los ficheros que se han ido añadiendo a cada tarea dentro de cada uno de los proyectos existentes. Los ficheros creados por cada tarea y subidos al servidor pueden ser aplicaciones, documentos, etc. Así mismo, se pueden crear estructuras de directorios.
- **Contacts:** Este módulo se utiliza para visualizar los contactos dentro y fuera de la organización, así como la creación de nuevos contactos. La cantidad de datos que se puede almacenar por cada contacto es considerable.
- **Forums:** Este módulo permite realizar las tareas típicas de un foro: creación de nuevos foros, tópicos, leer y responder mensajes, realizar búsquedas por parámetros, etc.

- **Departments:** Este módulo permite establecer los departamentos de nuestra organización, creando nuevos o editando los ya existentes. Después se puede visualizar que departamento esta realizando que proyectos.
- **Tickets:** Sistema que utiliza dotProject como helpDesk. Cada ticket representa un problema que se le ha presentado a un usuario del sistema. Tienen un estado de procesamiento como: Abiertos, Procesando, Cerrado y Eliminado. También tienen una prioridad: Baja, Normal, Alta y la más alta. Este módulo permite asignar tickets para que resuelvan las posibles incidencias que vayan surgiendo en un proyecto.
- **Users:** En este módulo están los usuarios que interactúan con el sistema. Están asociados a empresas y proyectos. Pueden tener diferentes roles dentro del sistema, variando su nivel de permisos en cada uno de ellos. Este módulo permite:
  - Visualizar el personal registrado en el dotProject.
  - Añadir, Editar, borrar y cambiar permisos de usuario.
  - Editar preferencias del usuario.

- **System:** Este módulo muestra todos elementos configurables que presenta dotProject, permite activar los gráficos de Gantt, integrar el dotProject a LDAP, establecer nombre, dominio, tiempo de duración de trabajo y días laborables. Las opciones más importantes son:
  - **Configuración del sistema:** Esta página es de gran importancia ya que será la encargada de configurar la parte básica del dotProject.
  - **Preferencias de Usuario Predeterminadas:** Estas opciones las podrá cambiar el usuario una vez autentificado.
  - **Editor de campos personalizados**: si se desea usar un campo que no viene por defecto para las empresas, proyectos, tareas o calendario se puede añadir con esta opción, modificando de esta forma dotProject sin necesidad de tocar una línea de código.

- **Admin:** Contiene la actividades relacionadas a la administración de usuarios, roles y configuración del sistema.
- **Resources:** Se trata de un módulo no estándar, por lo que para poder utilizarlo primero hay que descargarlo y añadirlo a la herramienta. Permite crear recursos no humanos (salas de reuniones, proyectores, etc.) y asignarlos a tareas. De momento no se permite vincularlo con el calendario.

2.4 Descripción de la base de datos del dotProject

La herramienta de gestión de proyectos dotProject posee características peculiares en cuanto al proceso de instalación de módulos a su núcleo principal.

Tanto en los módulos desarrollados por la comunidad como en los que son desarrollados por los desarrolladores que mantienen el sistema, las tablas pertenecientes al nuevo módulo se adhieren a la base de datos del sistema, no siempre guardando relación con la misma.

El dotProject utiliza una base de datos Mysql, esta última soporta distintas tecnologías de almacenamiento de datos, entre las que se destacan MyISAM e InnoDB, el primero de ellos (MyISAM) es el utilizado por el dotProject. En este motor de almacenamiento todos los datos se almacenan con el byte menor primero. Esto hace que sean independientes de la máquina y el sistema operativo, lo que nos brinda la opción de portabilidad de la base de datos, es decir, en caso que se desee instalar el dotProject en otro sistema operativo o en otra máquina y se desea conservar los datos, pues solo hacemos un backup a la base de datos y la importamos en la nueva. La única área de máquinas que pueden no soportar compatibilidad binaria son sistemas empotrados, que a veces tienen procesadores peculiares.

El dotProject inicialmente empezó funcionando con la version 4.x de MySQL, pero esta versión, así como todas las anteriores a ella no soportan ni procedimientos almacenados, ni triggers, por lo que todo lo relacionado con la conexión a la base de datos se maneja con la librería ADODB y la clase DBQuery que es

una capa de abstracción de la librería antes mencionada, razón por la que el dotProject no trabaja con procedimientos almacenados ni con triggers.

Resultaría bastante engorroso representar el modelo de datos correspondiente al dotProject, ya que el número de tablas del sistema base(sin la instalación de otros módulos auxiliares) asciende a 74 por lo que se propone la representación de las principales clases, es decir, las que aparecen en el modelo de dominio. (*ver figura 12*)

Por lo antes mencionado, describiremos las tablas representativas o las más importantes de la herramienta dotProject, estas tablas son:

- **companies** (*ver Anexo 5*)
- **projects** (*ver Anexo 6*)
- **departments** (*ver Anexo 7*)
- **projects_departments** (*ver Anexo 8*)
- **tasks** (*ver Anexo 9*)
- **tasks_departments** (*ver Anexo 10*)
- **users** (*ver Anexo 11*)
- **users_tasks** (*ver Anexo 12*)
- **files** (*ver Anexo 13*)

El diagrama de entidad-relación lo representaremos solo con las tablas y sus llaves primarias y foráneas, para que se pueda observar correctamente, y de esta forma se pueda comprender mejor las relaciones entre las tablas, pero contamos además con una ampliación de este diagrama donde se muestran todos los campos de las tablas en cuestión.(*ver Anexo 4*)

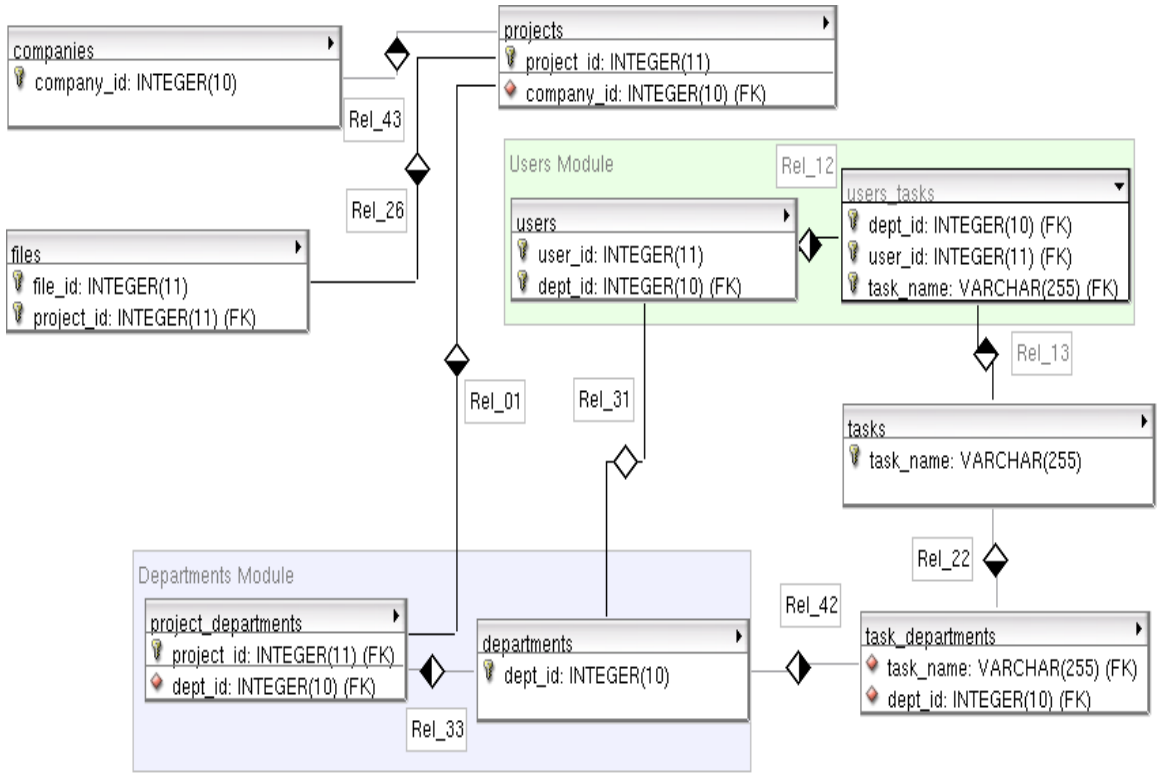


Figura 12. Diagrama de entidad-relación de las clases presentes en el modelo de dominio.

2.5 Estándar de codificación utilizado por dotProject

El estándar de codificación utilizado por el dotProject se inspira en el estándar de codificación de PEAR con ligeras modificaciones. Las principales razones para no tomar la norma de PEAR en su totalidad son las siguientes:

- PEAR es una librería, y dotProject es una aplicación.
- PEAR utiliza una licencia de librería, y dotProject utiliza una licencia de aplicación.
- PEAR está centrado en código ASCII, y el dotProject es un proyecto internacionalizado.

Básico

El código debe ser legible y estar escrito una sola vez, pero revisado y modificado varias veces. Lo cual permite que el código sea comprensible, y por tanto, que pueda ser mantenible. Aunque el rendimiento es importante, si hay una elección entre el rendimiento y la facilidad de lectura, la lectura debe tener preferencia.

Sangrado y longitud de la línea

La forma más elegante para marcar indentación (al principio del renglón) es la utilización de Tabs. Solamente se puede utilizar Tabs para la identación o sangrado. Cada nivel de identación puede ser solamente un Tab.

Estructuras de control

Estos incluyen **if, for, while, switch,** etc. A continuación se muestra un ejemplo de declaración, ya que es el más complicado de ellos:

```
<?php
    if ((condición1) || (condición2)){
        acción1;
     } elseif ((condición3) && (condición4)){
            acción2;
       } else {
            acción_por_defecto;
  }
?>
```

Las declaraciones de control deben tener un espacio entre las palabras clave de control y la apertura de paréntesis, para distinguirlos de las llamadas a funciones. Es recomendable utilizar siempre llaves, incluso en situaciones en las que son técnicamente opcional ya que estas aumentan la legibilidad y disminuye la probabilidad de introducir errores lógicos cuando se añaden nuevas líneas.

Funciones

Las funciones deben ser, sin espacios entre el nombre de la función, la apertura de paréntesis, y el primer parámetro, comas y espacios entre cada parámetro, y ningún espacio entre el último parámetro, el paréntesis de cierre, y el punto y coma. He aquí un ejemplo:

```
<?php
 $var = foo($bar, $variable, $pvar);
?>
```

Como se muestra en el ejemplo anterior, debe haber un espacio a ambos lados de un signo de igualdad utilizado para asignar el valor de retorno de una función a una variable. En el caso de un bloque de asignaciones, se puede introducir más espacio para facilitar la lectura e interpretación del código,
a continuación veremos un ejemplo:

```
<?php
 $short        = foo($bar);
 $long_variable = foo($variable);
?>
```

Definición de funciones

La declaración de funciones sigue el "estilo K & R", el cual permite ahorrar espacio vertical de lectura. Ejemplo:

```
<?php
   function fooFunction($arg1,$arg2){
      if (condición) {
            definiciones;
        }
      return $val;
    }
  ?>
```

Los argumentos de las funciones que tengan valores por defecto se colocan al final de la lista de argumentos que posee la función. Ejemplo:

```
<?php
    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;
     }
       ?>
```

Comentarios

Completar la documentación en línea con bloques de comentarios proporcionados. Estos bloques de comentarios deben seguir el estilo C (/ * * /) ó el estándar de C + + (/ /) .

Etiquetas de Código PHP

Siempre se debe usar **<?php** para abrir y **?>** para cerrar, y de esta forma delimitar el código PHP, nunca utilice <? >.

Etiquetas necesarias que tienen contenido variable

- **Descripciones Breves**

  Todos los bloques de comentarios deben ser breves descripciones, preferentemente en una oración pequeña.

- **Versiones de PHP**

  Uno de los siguientes bloques de comentarios debe ir en el nivel de página:

  - PHP version 4

- PHP version 5
- PHP versions 4 and 5

- **Autor**
  No existe ninguna norma para determinar cuándo un nuevo contribuyente de código debe añadirse a la lista de los autores de un determinado archivo fuente. En general, los cambios deben caer en la categoría sustancial, es decir, alrededor de 10% al 20% de los cambios en el código.

- **Ver**
  Añadir una etiqueta **@see** cuando quiera referirse a los usuarios o a otras secciones de la documentación del paquete.

Espaciamiento y Orden

Para facilitar la legibilidad a largo plazo del código fuente del dotProject, el texto y las etiquetas deberán ajustarse a la orden y el espacio proporcionado. Esta norma se adoptó a partir de la documentación estándar de Java.

Nombres convencionales

- **Clases**
  Las clases deben tener nombres descriptivos. Evite el uso de abreviaturas cuando sea posible. Los nombres de clase siempre debe comenzar con una letra mayúscula. Las clases fundamentales son con el prefijo C. El módulo principal de nombres de clase deben comenzar con el nombre del módulo. Ejemplos de buenos nombres de clase son los siguientes:
  - **CAppUI**
  - **MyModuleClass**

- **Métodos y funciones**

Las funciones tienen además del nombre del paquete un prefijo para evitar las colisiones entre los nombres de paquetes. La letra inicial del nombre (tras el prefijo) es minúscula, y cada letra que comienza una nueva "palabra" es mayúscula. Algunos ejemplos son:

- **Connect()**
- **GetData()**
- **BuildSomeWidget()**
- **XML_RPC_serializeData()**

Los miembros privados de la clase (es decir, los miembros de la clase que van a utilizarse sólo dentro de la misma clase en que se declaró;realmente PHP no soporta todavía ejecutar variables privadas) están precedidos por un solo subrayado. Por ejemplo:

- **_sort()**
- **_initTree()**
- **$this->_status**

En PHP5 los miembros protegidos de la clase (es decir, los miembros de la clase que van a utilizarse sólo dentro de la misma clase en la que se declaran o en subclases que se extienden desde la misma) no están precedidos por ningún subrayado. Por ejemplo:

- **protected $somevar**
- **protected function initTree()**

Todas las constantes deben ser siempre en mayúsculas y con subrayados para separar palabras. Estas poseen un prefijo de nombres constantes con el nombre de la clase o módulo en el que se utiliza. Las únicas constantes que se exceptúan de la norma de todas las mayúsculas y siempre deben estar en minúsculas son las constantes **true, false y null.**

Variables Globales

Si el módulo tiene que definir las variables globales, debe empezar su nombre en mayúsculas y con el nombre de la clase o del módulo.

2.6 Guía para la creación de un módulo

El dotProject es de código abierto, lo cual nos permite reutilizar el código y reimplementar funcionalidades ya existentes o implementar nuevas de acuerdo a las necesidades del usuario. A continuación veremos los pasos a seguir para la creación de módulos, esto nos dará la posibilidad de tener mayor organización y una homogeneidad a la hora de crear nuevos módulos.

Lo primero que haremos será crear un directorio con el nombre del módulo. Las imágenes e iconos se deben guardar en un directorio **/images** dentro de ese mismo directorio. Luego se debe desarrollar un script de instalación (**setup.php**) para instalar,actualizar,suprimir y configurar el módulo.(*ver Anexo 2*)

El significado de cada campo es bastante intuitivo. A la hora de nombrar y renombrar variables se recomienda por claridad utilizar el mismo criterio que el usado habitualmente en otros módulos. Esto además ayuda a encontrar clases, atributos y demás "por intuición" , además es utilizado en algunas situaciones por el dotProject para interpretar el contenido de los elementos del módulo. También es aplicable a la hora de nombrar otros elementos del código (clases, variables, atributos de campos en SQL, etc).

Además del archivo de instalación del módulo es necesario un archivo para implementar la clase del módulo. Este debe tener su nombre en el nombre del módulo y el sufijo "**.class**" y la extensión ".**php**".(*ver Anexo 3*)

El archivo **setup.php** es una petición global del módulo **dp,** donde se encuentra la declaración de la clase CDpObject que se extiende por la clase para acceder a CsprintBacklog, para acceder a la tabla sprint_backlog de la base de datos. Todos los atributos de la tabla son colocados como atributos de la clase para tener una comunicación entre las dos entidades. El único método de escritura en la clase CsprintBacklog es el método constructor, el cual le informa a la clase cuál es el nombre de la tabla y la clave principal en la base de datos.

Todavía hay tres archivos necesarios para el desarrollo del módulo. El primero de estos archivos se utiliza para hacer las secuencias de scripts para comunicarse con la base de datos, para la integración, modificación y supresión de los registros del módulo **(do_modulo_aed.php)**. Los dos últimos se utilizan para diseñar la interfaz de usuario; uno de ellos es el archivo **addedit.php** que se utiliza para insertar o modificar registros y el otro es el **index.php** que se utiliza como interfaz inicial del módulo para listar, filtrar, etc. Aunque los módulos pueden contener otros archivos, estos son son la base para la construcción de un módulo.

Algo muy importante y que se debe tener en cuenta a la hora de crear un módulo es que se debe crear su correspondiente fichero de traducción al español. Como se indica en los ficheros de traducción, se debe usar obligatoriamente el sistema de traducciones del módulo de administración de dotProject, nunca editarlos a mano. Los ficheros de traducción se encuentran en el directorio **/locales**, con versiones en Inglés (/en)-que es la que toman como referencia las demás traducciones- y en todos los idiomas que nos hayamos descargado (en principio, sólo Español (/es)).

Cambiar el nombre del proyecto implica cambiar el nombre de la clase que usaremos para realizar nuestras consultas a la base de datos. El dotProject carga al inicio una clase de nombre nombreMódulo.class.php, a través de la que podremos asignar atributos a cada tabla.

En el dotProject todos los módulos siguen un cierto convenio a la hora de asignar nombres a variables y valores de la base de datos, es por ello que el dotProject cuenta con un convenio de nombres el cual facilita la comprensión y la portabilidad del código y además nos permite deducir nombres de variables sin necesidad de recurrir a la documentación.

Convenio de nombres

A continuación mostraremos algunos de ellos:

- **Ficheros**

- **addobjeto.php** o **addedit.php** para añadir o editar objetos.
- **view.php** para ver el objeto más importante del módulo.
- **vw_pestaña.php** para cada pestaña visible desde el index del módulo.
- **index.html** que será el que defina cuándo y cómo ver cada pestaña.
- **do_módulo_aed.php** para definir la interfaz con la base de datos.
- **módulo.class.php**, se explica a continuación.

- **Clases**
  - Se declaran en un fichero módulo.class.php, si bien en **setup.php** se encuentra la sentencia de creación de la tabla.
  - La clase de configuración/instalación/desinstalación en **setup.php** se llamará SetupObjetos, respetando las mayúsculas y el nombre en plural.
  - Cada atributo definido en la clase Objetos se llamará objeto_atributo, siendo objeto_id su identificador principal. En caso de que el atributo haga referencia a otra tabla se escribirá como objeto_tabla, con el nombre de la tabla en singular.

- **Tablas de la base de datos**

  Se sigue el mismo convenio que se acaba de describir para las clases: Nombre de tabla en plural y atributos con nombre de tabla en singular seguidos de guión bajo y nombre de atributo.

- **Funciones javascript**
  - Las funciones tipo **Submit** serán llamadas **SubmitIt()**.
  - Las funciones tipo **Delete** serán llamadas **DelIt().**
  - Las funciones que abran ventanas tipo pop up se llamarán popNombreVentana().

- Las funciones para asignar valores de vuelta de ventanas tipo **pop up** se llamarán setNombreVentana().

Al crear un módulo vemos que no es necesario asignar un tipo a los atributos, porque el tipo se hereda de la base de datos. Las funciones **CTabla**() y **delete**() son el constructor y destructor de cada objeto que insertemos/eliminemos de la base de datos. En el constructor no será necesario cambiar nada más que el renombrado, pero en el destructor puede que necesitemos añadir algún **delete** extra si nuestras tablas tienen campos multivaluados.

Existe otro convenio de nombres a la hora de dar valores a los campos de elementos en MySQL. Las tablas normalmente se denominarán en plural y cada atributo de cada elemento de la misma con el nombre de la tabla en singular seguido del nombre del atributo. Esto será extremadamente útil para deducir los atributos de otras tablas sin necesidad de recurrir a la documentación de dotProject. Por ejemplo project_name, task_name, company_name, etc, o vínculos entre tablas como task_project, etc.

El siguiente paso es realizar las operaciones de creación de tablas en la base de datos. Para ello editaremos el fichero setup.php que se ejecutará al instalar o desinstalar el módulo. Por tanto, si realizamos modificaciones en la base de datos durante el proceso de desarrollo del módulo, debemos desinstalarlo, modificar el código en todos los ficheros implicados, y reinstalarlo para que los cambios tengan efecto (aunque también es posible editar la base de datos por separado...). Como la creación y destrucción de tablas se lleva a cabo a través de consultas en MySQL, es relativamente sencillo crear tablas muy complejas o con elementos cruzados entre ellas.

Todo módulo debe contener una clase que por "convenio dotProject" se llama CsetupMódulo, cuyo código contiene:

- **Una función de instalación** en la que se crean las tablas usando código SQL.

```
$sql = "CREATE TABLE  nombre_tabla ( "
                " nombre_tabla _id int(11) unsigned NOT NULL auto_increment"
```

```
                ", nombre_tabla _concepto text"
                ", nombre_tabla _valor decimal(10,2) default '0.00'"
                ", nombre_tabla _proyecto int(11)"
                ", nombre_tabla _tarea int(10)"
                 ", PRIMARY KEY(nombre_tabla _id)"
                ", UNIQUE KEY(nombre_tabla _id)"
               ", FOREIGN KEY(nombre_tabla _proyecto)  REFERENCES    projects(project_id)"
               ", FOREIGN KEY(coste_tarea) REFERENCES tasks(task_id)"
             ") TYPE=MyISAM;";

            db_exec( $sql ); db_error();
            return null;
        }
```

- **Una función de desinstalación**, en la que se eliminan las tablas SQL.

```
function remove() {
    db_exec( "DROP TABLE costes;" );
    db_exec( "DROP TABLE tarifas;" );
  return null;
 }
```

Cómo cambiar el directorio de un módulo

Esto sólo es aplicable a los módulos que no vienen con la distribución original de dotProject. Para que todo funcione correctamente basta con coger uno de estos módulos y hacer lo siguiente:

1. Cambiar en el **setup.php** el contenido de $config['mod_directory'].
2. Renombrar todos los "m=módulo_proyecto" con el nombre del nuevo directorio.

Definiciones de las funciones y los métodos de dotProject

El dotProject tiene un conjunto de funciones y métodos que se utilizan para ayudar a los módulos con la función de alcance global. Estas funciones son en realidad las llamadas a otros métodos y funciones para

las bibliotecas de los niveles más bajos de dotProject. De esta forma se muestran los métodos y funciones de interés para construir un nuevo módulo. (*ver anexo 1*)

2.7 Propuesta de un módulo que permita la creación de nuevos módulos

A partir del análisis y estudio de la estructura genérica que presentan los módulos de dotProject se determinó que la misma no brinda muchas facilidades a los desarrolladores. Producto a que no se puede lograr una verdadera reutilización de los componentes que posee y que resulta difícil realizar el mantenimiento de los mismos es necesario optar por otra estructura que elimine estas y otras deficiencias que posee la ya existente.

La herramienta dotProject constituye un framework por todas las clases de acceso a datos y lógica del negocio que utilizan los módulos que brinda esta herramienta en los directorios **class** y **lib** que fueron descritos anteriormente. Partiendo de esta enorme ventaja sería considerablemente útil hacer uso del patrón arquitectónico MVC para la construcción de los nuevos módulos de dotProject porque se contaría desde el inicio con todos los componentes el Modelo y los desarrolladores solo tendrían que concentrarse en las diferentes vistas que tenga el módulo y en la construcción de los controladores que actúen como intermediarios con el Modelo.

Este patrón arquitectónico que se propone para la creación de nuevos módulos para el dotProject nos permite desarrollar un software muy mantenible, supone un diseño muy poco acoplado, lo que favorece la reutilización de código. Además la lógica relacionada con los datos se incluye en el **modelo**, el código de la presentación en la **vista** y la lógica de la aplicación en el **controlador**.

El patrón arquitectónico MVC separa la lógica de negocio (**el modelo**) y la presentación (**la vista**) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. Si por ejemplo una misma aplicación debe ejecutarse tanto en un navegador estándar como un un navegador de un dispositivo móvil, solamente es necesario crear una vista nueva para cada dispositivo; manteniendo el controlador y el modelo original. **El**

controlador se encarga de aislar al **modelo** y a la **vista** de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email, etc.). **El modelo** se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación (Symfony, la guía definitiva 2009).

Como anexo de esta investigación se entrega un nuevo módulo (**Creator**) para la herramienta dotProject. Este módulo permite crear los componentes genéricos que son imprescindibles para el correcto funcionamiento de un módulo. En la figura 13 se muestra un ejemplo de la nueva estructura que presentarán los módulos que sean creados con el mismo.

Figura 13. Propuesta de la nueva estructura del módulo del dotProject

- El fichero **index.php** se utiliza como interfaz inicial del módulo para listar, filtrar, etc.
- El fichero **setup.php** es un script para instalar, actualizar, eliminar y configurar el módulo.
- El directorio **includes** pertenece a la parte del **Modelo,** ya que este último es el encargado de todas las clases de acceso a datos y de la lógica de negocio que utilice el módulo.

- El directorio **view** pertenece a la parte de la **Vista,** ya que esta última es la responsable de recibir los datos del **modelo** y mostrarlos al usuario. Cada una de las vistas por lo general tiene asociado un controlador según la funcionalidad que realiza.
- En el directorio **controller** se encuentran todos los ficheros responsables de recibir desde la vista los eventos de entrada y hacer las peticiones al modelo.

Pasos para crear un módulo utilizando el módulo "Creator".

Para crear un nuevo módulo primero entramos el nombre deseado para dicho módulo. (*ver Anexo 14*) Luego debemos modificar los ficheros **setup.php** e **index.php** y adaptarlos a nuestras necesidades, para ello se cuenta con una explicación detallada del significado y el valor de cada variable, lo cual servirá de guía a la hora de editar dichos ficheros. Conjuntamente con la configuración de estos dos ficheros, se elige el icono que mostrará el módulo y presionamos el botón "Enviar".(*ver Anexo 15*)

Si todos los datos están correctos se creará nuestro módulo, de lo contrario debemos volver a la página principal y repetir nuevamente todo el proceso de creación del módulo.

Una vez creado el módulo, procedemos al proceso de instalación del mismo, para ello, nos aseguramos que se ha creado nuestro módulo en el directorio **/modules,** luego tenemos que activar el módulo creado, para ello el dotproject brinda una interfaz bastante amigable con el usuario. La ruta a seguir para activar los módulos es la siguiente:

- Sistema
- Ver módulos
- Seleccionar módulo a instalar
- Activar el módulo.
- Darle visibilidad módulo (para que aparezca en el menú general)

2.8 Conclusiones Parciales

En este capítulo se ha definido el modelo de dominio y la estructura de directorios del dotProject, así como una guía para la creación de módulos para esta herramienta. Se realiza la propuesta de un módulo para generar nuevos módulos con las características arquitectónicas del patrón MVC -donde se tuvo en cuenta el estándar de codificación del dotProject-. Conjuntamente se realiza la descripción de la arquitectura y la descripción de la base de datos de la herramienta dotProject. Además se explicó como se aplica el patrón de arquitectura en capas para dar solución del problema planteado. Finalmente quedó establecida la descripción de la arquitectura de la herramienta dotProject.

Conclusiones Generales

La gestión de proyectos es de suma importancia, ya que contribuye a la organización de los proyectos de software. Los sistemas de gestión humanizan y permiten una automatización de las tareas de gestión logrando de forma automática aumentar el numero de elementos a tener en cuenta para la toma de decisiones.

El dotProject -herramienta libre propuesta para la gestión de proyectos- posee las características técnicas y desde el punto de vista de herramienta de gestión para convertirse en una excelente aplicación, se cuenta con diferentes diagramas que servirán como material de apoyo para una mayor comprensión del funcionamiento y la arquitectura de esta herramienta.

- Se cumplieron todos los objetivos trazados en esta investigación y a la vez se cumplió con el problema científico a partir de la realización de la descripción de la arquitectura de la herramienta dotProject.
- Es de vital importancia que todo sistema de software esté respaldado por una arquitectura sólida que facilite el entendimiento del mismo, que sea capaz de organizar el desarrollo, fomentar la reutilización y hacer evolucionar el sistema.
- Se realizó un estudio profundo acerca de varios elementos arquitectónicos que existen en la actualidad, para hacer uso de las mejores técnicas de diseño arquitectónico.
- Se realizó una profunda investigación de las funcionalidades y estructura de directorios del dotProject, arribando a la conclusión de que el dotProject presenta una arquitectura de tres capas.
- Se definieron componentes y funcionalidades reusables que fueron empleados, permitiendo acortar el tiempo de desarrollo de la aplicación.

Recomendaciones

Con el propósito de ampliar y mejorar la documentación de la arquitectura presentada en este trabajo,se proponen las siguientes recomendaciones:

- Efectuar un continuo refinamiento de la arquitectura de software del dotProject.

- Que se comience a utilizar el módulo “Creator” para la confección de nuevos módulos.

- Comenzar a utilizar el presente trabajo en aquellos proyectos productivos que utilicen el dotProject como herramienta para la gestión de proyectos con el fin de comprender la arquitectura de dicha herramienta y de esta forma poder desarrollar correctamente nuevas funcionalidades.

Referencias Bibliográficas

C.Mauricio. Patrón (MVC) Sugerido | OrfeoGPL. 2007. [cited 19 May 2009]. Available from world wide web: <http://orfeogpl.org/ata/node/242>.

Joaquín Gracia. Patrones de diseño. Análisis y Diseño. Ingeniería del Software. 2005. [cited 19 May 2009]. Available from world wide web: <http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php>.

Juan B. OpenProj, una opción como herramienta de planificación PymeCrunch. 2008. [cited 19 May 2009]. Available from world wide web: <http://pymecrunch.com/openproj-una-opcion-como-herramienta-de-planificacion>.

Luis A. Guerrero. Luis A. Guerrero - CC40B. 2009. [cited 19 May 2009]. Available from world wide web: <http://www.dcc.uchile.cl/~luguerre/cc40b/clase6.html>.

maestrosdelweb.com. ¿Qué es Smarty? 2007. [cited 19 May 2009]. Available from world wide web: <http://www.maestrosdelweb.com/editorial/que-es-smarty/>.

Mario Alberto Arredondo Guzmán. JpGraph - OO Graph Library for PHP. 2009. [cited 19 May 2009]. Available from world wide web: <http://manuales.dgsca.unam.mx/programasphp/jpgraph.html>.

Miguel Angel Alvarez. Qué son los Gestores de Proyectos. 2007. [cited 19 May 2009]. Available from world wide web: <http://www.desarrolloweb.com/articulos/que-son-gestores-proyectos.html>.

Paul Clemens. Frameworks y arquitecturas de software. 2008. [cited 19 May 2009]. Available from world wide web: <http://www.mailxmail.com/curso-programacion-avanzada/frameworks-arquitecturas-software>.

Referencias Bibliográficas

Rodrigo Escárate. Revista Gerencia. 2007. [cited 19 May 2009]. Available from world wide web: <http://www.emb.cl/gerencia/articulo.mv?sec=12&num=150>.

www.sistemasdigitales.com.mx. Administración de Proyectos, Project Management, PMP, Mexico. 2008. Available from world wide web:<http://www.sistemasdigitales.com.mx/paginas/proyectos.html>.

Carlos Reynoso. Si Saber No Es Un Derecho, Seguro Será Un Izquierdo. En Linea 1031Desarrolladores. 2004. [cited 19 de Mayo 2009].Available from world wid web:<http://www.willydev.net/ >

Craig Larman. UML y Patrones: una Introducción al Análisis y Diseño Orientado a Objetos y al Proceso Unificado, Segunda Edición, Tomo I Capítulo 18. 2003.[cited 20 de mayo 2009]

Jesus Villamor. Introducción a los sitemas de información. 2009. [cited 27 May 2009]. Available from world wide web: <http://www.it.uc3m.es/mcfp/docencia/si/material/1_cli-ser_mcfp.pdf>.

R.Niella.Capítulo1:Cliente Servidor.2009.[cited 25 May 2009].Available from world wide web: <http://ar.geocities.com/r_niella/Document/t_cap1.htm>.

www.visual-paradigm.com. UML CASE Tools - Free for Learning UML, Cost-Effective for Business Solutions.2009. [cited 27 May 2009]. Available from world wide web: <http://www.visual-paradigm.com/product/vpuml/>.

Eric L. Laffoon.Quanta Plus Web Development tool.2004.[cited 5 June 2009]. Available from world wide web: <http://quanta.sourceforge.net/>.

temariotic.wikidot.com. temariotic: La arquitectura cliente servidor. 2009. [cited 5 June 2009]. Available from world wide web: <http://temariotic.wikidot.com/la-arquitectura-cliente-servidor>.

Referencias Bibliográficas

www.php.net/. PHP: ¿Qué se puede hacer con PHP? - Manual. 2009. [cited 5 June 2009]. Available from world wide web: **<http://www.php.net/manual/es/intro-whatcando.php>.**

www.microsoft.com. Microsoft MSDN - Mediacenter. 2009. [cited 5 June 2009]. Available from world wide web: **<http://www.microsoft.com/Spanish/msdn/latam/mediacenter/webcast/architect.aspx>**.

Symfony, la guía definitiva. El patrón MVC | Symfony 1.1, la guía definitiva | LibrosWeb.es. 2009. [cited 18 May 2009]. Available from world wide web: **<**http://www.librosweb.es/symfony_1_1/capitulo2/el_patron_mvc.html>.

Bibliografía

Paulo Nunes. Concepto de Gestión de Proyectos. 2008. [cited 19 May 2009]. Available from world wide web: <http://www.knoow.net/es/cieeconcom/gestion/gestiondeproyectos.htm>.

Miguel M. Anastacio Velásquez. http://www.informatizate.net - Model View Controller (MVC). [cited 19 May 2009]. Available from world wide web:
<http://www.informatizate.net/articulos/model_view_controller_mvc_20040324.html>.

Ernesto Serrano. dotProject para planificación de proyectos | Abartia Team. 2009. [cited 13 May 2009]. Available from world wide web: <http://www.abartiateam.com/dotproject>.

Luis A. Guerrero."CC40B-Análisis y Diseño Orientado a Objetos".[cited 20 de abril 2009].
Available from world wide web: <http://www.dcc.uchile.cl/~luguerre/cc40b/clase6.html.>

Pressman, Roger S. "Ingeniería de Software. Un enfoque práctico." 5ta Edición (traducción de la edición original en Inglés "Software Engineering. A practical approach"), Editorial Mac Graw Hill, Madrid, España.

Shaw, M. y Garlan, D. "Software Architecture". Editorial Prentic-Hall, New York, E.U.A. 1996.

Gracia, Joaquín (2005) .Patrones de diseño. Available from world wide web:
< http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php>

ANÓNIMO. Estimación de proyectos de software, 2007. [Febrero, 2009]. Available from world wide web:
<http://members.fortunecity.com/patriciob/estimacion.htm>

Bibliografía

FREITAG, C. S. Y. K. Sitio oficial de la herramienta TaskJuggler, 2008. [Marzo, 2008]. Available from world wide web:< http://www.taskjuggler.org>

TEAM, A. Gestión de proyectos con dotProject, 2007. [Marzo, 2009]. Available from world wide web: <http://www.abartiateam.com/dotproject.html **>**

Sitio oficial de la herramienta B-kin, 2008. [Marzo, 2009]. Available from world wide web: <http://www.b-kin.com>

Sitio oficial de la herramienta Redmine, 2008. [Marzo, 2009]. Available from world wide web: <http://www.redmine.org>

Sitio oficial de la herramienta Basecamp, 2008. [Marzo, 2009]. Available from world wide web: < http://www.basecamphq.com>

Sitio oficial de la herramienta ActiveCollab, 2008. [Marzo, 2009]. Available from world wide web: <http://www.activecollab.com>

Glosario de Términos

Estilo K&R: es el estilo más usado en el lenguaje C y PHP. Se trata de abrir la llave en la misma línea de declaración de la orden, indentando los siguientes pasos al mismo nivel que la llave y cerrando la llave en el mismo nivel que la declaración.

Framework: es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soportes de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

HTML: es un conjunto de símbolos o palabras que definen varios componentes de un documento Web; estos se pueden ver siempre dentro de las etiquetas "<", ">". HiperText Markup Languaje es el nombre que estas siglas representan, creado por Tim Berners-Lee en 1991.

Identación: Por indentación se entiende mover un bloque de texto hacia la derecha insertando espacios o tabuladores para separarlo del texto adyacente.

PDF: Formato gráfico creado por la empresa Adobe que reproduce cualquier tipo de documento en forma digital idéntica, permitiendo a las distribuciones electrónica de los mismos a través de la red.

Proyecto: Es el elemento organizativo a través del cual se gestiona el desarrollo de software y el resultado del mismo es una versión de un producto.

Producto: Artefactos que se crean durante el transcurso de la vida el proyecto.

Proceso: es un conjunto de actividades necesarias para transformar los requisitos de los usuarios en un producto.

Glosario de Términos

LDAP: es un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red. También es considerado base de datos (aunque su sistema de almacenamiento puede ser diferente) a la que pueden realizarse consultas.

Tabs: es la abreviatura de Tabulador. Tabular significa poner algo en forma de tabla.

MySQL: Sistema de gestión de base de datos relacional, multihilo y multiusuario.

PHP: Significa " PHP Hypertext Pre-processor " (inicialmente PHP Tools, o, Personal Home Page Tools), y se trata de un lenguaje interpretado usado para la creación de aplicaciones para servidores, o creación de contenido dinámico para sitios Web. Últimamente también para la creación de otro tipo de programas incluyendo aplicaciones con interfaz gráfica usando las librerías Qt o GTK+.

Diagramas de Gantt: es una popular herramienta gráfica cuyo objetivo es el de mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado. A pesar de que, en principio, el diagrama de Gantt no indica las relaciones existentes entre actividades, la posición de cada tarea a lo largo del tiempo hace que se puedan identificar dichas relaciones e interdependencias.

Gestión de Proyecto: concepto que describe varios tipos de software, incluyendo programación, asignación de recursos, software de colaboración, comunicación y sistemas de documentación, utilizados para ayudar a organizar un proyecto complejo en diferentes tareas y en un tiempo determinado.

Sistema empotrado: es un sistema informático (hardware + software) de tiempo real integrado en un sistema de ingeniería más general, en el que realiza funciones de control, procesamiento y/o monitorización.

Anexos

Anexo 1 Funciones y métodos del dotProject.

Nombre de la Función o Método	Fichero /Archivo	Descripción
bool **getDenyEdit**(string $m, int $item_id)	**config.php**	Devuelve **true** si el usuario tiene derecho a **escribir** en el módulo $m o en su $item_id , de lo contrario devuelve **false**.
bool **getDenyRead**(string $m, int $item_id)	**config.php**	Lo mismo que el anterior,pero de **solo lectura**.
db_bindHashToObject(array $hash, object $object, string $prefix, bool $checkslashes, bool $bindAll)	**db_connect.php**	Conecta el contenido de $hash con las propiedades del objeto $object.
db_delete(string $table, string $keyName, mixed $keyValue)	**db_connect.php**	Excluye las líneas de la tabla **$table** donde el valor de la variable $KeyName es $keyValue.
array db_loadList(string $sql, int $maxrows)	**db_connect.php**	Devuelve una matriz asociativa de la sentencia SQL, limitada por la variable $maxrows.
array db_loadObject(string $sql,object &$object, bool $bindAll)	**db_connect.php**	Liga el contenido de la primera línea de sentencia SQL con el objeto $object.
string addHistory(string $description, int $project_id, int $module_id)	**main_functions.php**	Agrega una entrada en el historial de cambio del recurso marcado.
string defVal(mixed $var, mixed $def)	**main_functions.php**	Devuelve el valor por defecto $def si la variable $Var no es setada.
string dPformSafe(string $txt, bool $deslash)	**main_functions.php**	Convierte la variable $txt para una cadena con comillas doble al principio y al final.
string dPgetParam(array &$arr,string $name, mixed $def)	**main_functions.php**	Devuelve el nombre de una matriz, el cual por defecto es el valor de la variable $def.
string arraySelect(array &$arr, string $select_name, array $select_attribs, string $selected, bool $translate =false)	**main_functions.php**	Devuelve una forma de Lenguaje de marcado de hipertexto (HTML) basado en la clave del 1er arreglo, si la variable $translate es igual a **true**.

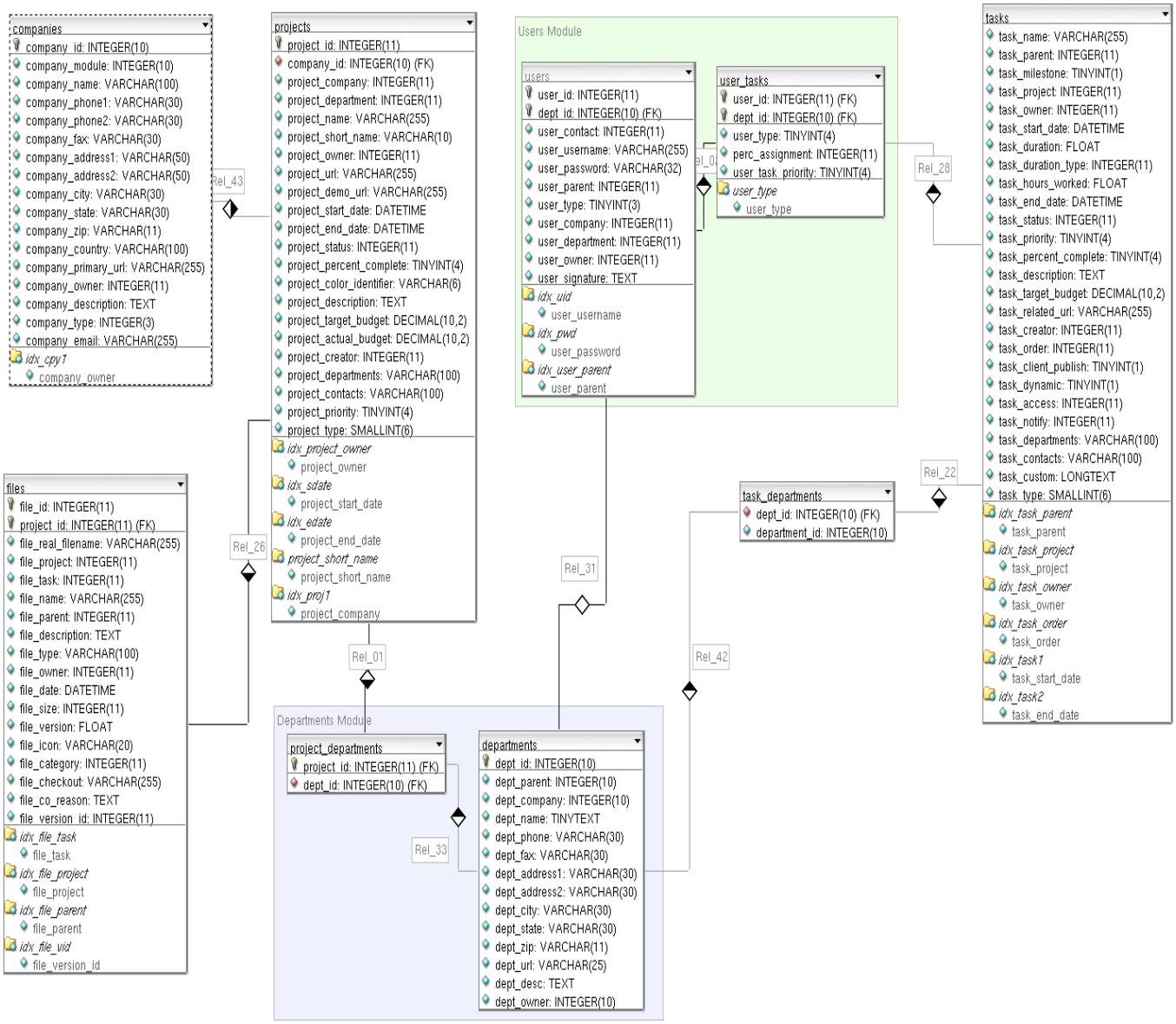
Anexo 2 Script para crear un módulo en el framework dotProject

```
<?php
 $config = array();
 $config['mod_name'] = 'Prueba'; // nombre del módulo
 $config['mod_version'] = '0.1'; // especificar la versión del módulo
 $config['mod_directory'] = 'prueba'; //define donde esta el directorio del módulo
 $config['mod_setup_class'] = 'DP_Prueba';//el nombre de la clase instaladora que se define más adelante en
este fichero
 $config['mod_type'] = 'user'; //tipo de módulo, como estándar se pone 'user' para los módulos adicionales
 $config['mod_ui_icon'] = 'prueba.jpg'; // nombre del icono o logo del módulo
 $config['mod_description'] = 'Esto es un módulo de prueba'; // Pequeña descripción del módulo
 $config['mod_config'] = false; // determina si se muestra o no el link de configuración en la vista de
administración del módulo.

 if (@$a == 'setup')
    {
     echo dPshowModuleConfig( $config );
    }

 class DP_Autos
 {
  function install()
  {
  /*En esta función se especifican los requerimientos del nuevo módulo, desde las tablas que necesite hasta la
          creación de directorios auxiliares*/
       //ejemplo de la creación de una tabla para el módulo
 $sql = '(
             prueba_id VARCHAR(45) NOT NULL,
             descripcion TEXT NOT NULL,
             PRIMARY KEY(prueba_id)
            )TYPE=MyISAM';

 $q = new DBQuery;
 $q->createTable('prueba');
 $q->createDefinition($sql);
 $q->exec();
 }

 function remove()
 {
 /*forma de eliminar los recursos creados por el módulos y que eran necesarios solo para su  funcionamiento*/
 //como eliminar la tabla del módulo
  $q = new DBQuery;
  $q->dropTable('prueba');
  $q->exec();
 }
}
?>
```

Anexo 3 Script de especificación de la clase CsprintBacklog.

```
<?php
    require_once( $AppUI->getSystemClass ('dp' ) );
     class CSprintBacklog extends CDpObject {
     var $sprint_backlog_id = NULL;
     var $product_backlog_id = NULL;
     var $sprint_backlog_nm = NULL;
     var $sprint_backlog_ds = NULL;
     var $sprint_backlog_st = NULL;
     var $requisitor_id = NULL;
     var $user_id = NULL;
     var $hours_1 = NULL;
     var $hours_2 = NULL;
     var $hours_3 = NULL;
     var $hours_4 = NULL;

     function CsprintBacklog()
      {
       $this->CDpObject( 'sprint_backlog', 'sprint_backlog_id' );
      }
    }
?>
```

Anexo 4 Diagrama de entidad-relación del modelo de dominio del dotProject.



Anexo 5 Descripción de la tabla **companies**

companies: Tabla que almacena las características de la(s) empresa(s).

Campo	Tipo	Nulo	Predeterminado	Comentarios
company_id	int(10)	No		Identificador de la empresa
company_module	int(10)	No	0	Identifica el ide del módulo empresa
company_name	varchar(100)	Yes		Nombre de la empresa
company_phone1	varchar(30)	Yes		Teléfono 1 de la empresa
company_phone2	varchar(30)	Yes		Teléfono 2 de la empresa
company_fax	varchar(30)	Yes		Fax de la empresa
company_address1	varchar(50)	Yes		Dirección 1 de la empresa
company_address2	varchar(50)	Yes		Dirección 2 de la empresa
company_city	varchar(30)	Yes		Muestra la ciudad a la que pertenece la empresa
company_state	varchar(30)	Yes		País o estado al que pertenece la empresa
company_zip	varchar(11)	Yes		Muestra el código postal de la empresa.
company_country	varchar(100)	Yes		País donde se encuentra la empresa.
company_primary_url	varchar(255)	Yes		Dirección web de la pagina de la empresa
company_owner	int(11)	No		Identifica el creador de la empresa.
company_description	text	No	0	Texto descriptivo de la empresa.
company_type	int(3)	No		Identifica el tipo de empresa.
company_email	varchar(255)	Yes	Null	Muestra la dirección de correo de la empresa.

Anexo 6 Descripción de la tabla **projects**

projects: Tabla que contiene los datos de los proyectos

Campo	Tipo	Nulo	Predeteminado	Comentarios
project_id	int(11)	No		Identifica el id del proyecto.
project_company	int(11)	No	0	Identifica la empresa a la que pertenece el proyecto
project_department	int(11)	No	0	Identifica el departamento al que pertenece el proyecto
project_name	varchar(255)	Si	Null	Nombre del proyecto
project_short_name	varchar(10)	Si	Null	Alias del proyecto
project_owner	int(11)	Si	0	Identifica el id del usuario que creó el proyecto
project_url	varchar(255)	Si	Null	Dirección web del proyecto
project_start_date	datetime	Si	Null	Fecha de inicio del proyecto.
project_end_date	datetime	Si	Null	Identifica la fecha en que se estima que deba terminar el proyecto.
project_status	int(11)	Si	0	Identifica el id del estado en que se encuentra el proyecto
project_percent_complete	tinyint(4)	Si	0	Muestra el porciento completado del proyecto.
project_color_identifier	varchar(6)	Si	eeeeee	Identifica el color que posee el proyecto
project_description	text	Si	Null	Muestra una descripción del proyecto
project_target_budget	decimal(10,2)	Si	0.00	Muestra el presupuesto tentativo del proyecto
project_actual_budget	decimal(10,2)	Si	0.00	Muestra el presupuesto real del proyecto

Anexo 6 Descripción de la tabla **projects** (Continuación)

project_creator	int(11)	Si	0	Identifica el id del creador del proyecto
project_departments	char(100)	Si	Null	Muestra el id del o los departamentos que forman parte del proyecto
project_contacts	char(100)	Si	Null	Muestra el ide de los contactos del proyecto
project_priority	tinyint(4)	Si	0	Muestra el id de la prioridad del proyecto
project_type	smallint(6)	No	0	Muestra el id del tipo de proyecto

Anexo 7 Descripción de la tabla **departments** (Continuación)

departments: Tabla que almacena los datos de los departamentos.

Campo	Tipo	Nulo	Predeterminado	Comentarios
dept_id	int(10)	No		Identificador del departamento.
dept_parent	int(10)	No	0	Identificador del id del departamento padre,esto es si el departamento es derivado de algún departamento.
dept_company	int(10)	No	0	Identificador del id de la empresa a la que pertenece el departamento
dept_name	tinytext()	No	Null	Nombre del departamento
dept_phone	varchar(30)	Si	Null	Número de teléfono del departamento.
dept_fax	varchar(30)	Si	Null	Número de fax del departamento.
dept_address1	varchar(30)	Si	Null	Dirección del departamento.
dept_address2	varchar(30)	Si	Null	Continuación de la Dirección del departamento o la continuación de la primera dirección

Anexo 7 Descripción de la tabla **departments** (Continuación)

dept_city	varchar(30)	Si	Null	Ciudad donde se encuentra el departamento
dept_state	varchar(30)	Si	Null	Estado o país donde radica el departamento
dept_zip	varchar(11)	Si	Null	Identificador del código postal de la ciudad donde se encuentra la empresa
dept_url	varchar(25)	Si	Null	Dirección Web de la empresa
dept_desc	text	Si	Null	Descripción del departamento.
dept_owner	int(10)	No	0	Identificador del dueño o creador del departamento

Anexo 8 Descripción de la tabla **projects_departments**

projects_departments: Tabla que surge por la relación de muchos a muchos (**n*n**) que se establece entre la tabla **projects** y la tabla **departments.**

Campo	Tipo	Nulo	Predeterminado	Comentarios
project_id	int(10)	No		Identificador del proyecto.
dept_id	int(10)	No		Identificador del departamento que agrupa a otros departamentos.

Anexo 9 Descripción de la tabla **tasks**

tasks: Tabla que almacena todos los datos de las tareas.

Campo	Tipo	Nulo	Predeterminado	Comentarios
task_id	int(11)	No		Identifica la tarea
task_name	varchar(255)	Si	Null	Nombre de la tarea
task_parent	int(11)	Si	0	Identificador para la tarea principal,esto es en el caso de que existan varias subtareas.
tasks_project	int(11)	No	0	Identificador del proyecto al que pertenece la tarea
task_owner	int(11)	No	0	Identifica el dueño de la tarea
task_start_date	datetime	Si	Null	Muestra la fecha de inicio de la tarea
task_duration	float	Si	0	Contiene la cantidad de horas de duración de la tarea
task_hours_worked	float	Si	0	Muestra el tiempo trabajado
task_end_date	datetime	Si	0	Muestra la fecha de fin de la tarea
task_status	int(11)	Si	0	Muestra el estado en que se encuentra la tarea
task_priority	tinyint(4)	Si	0	Muestra la prioridad de la tarea
task_percent_complete	tinyint(4)	Si	0	Muestra el progreso -en porciento (%)- de la tarea
task_description	text	Si	Null	Descripción de la tarea
task_creator	int(11)	No	0	Identifica el creador de la tarea
task_access	int(11)	No	0	Identifica el tipo de acceso a la tarea
task_notify	int(11)	No	0	Identifica si la tarea será notificada o no

task_departments	char(100)	Si	Null	Muestra el nombre de los departamentos asociados a esta tarea
task_contacts	char(100)	Si	Null	Muestra el nombre de los contactos asociados con la tarea
task_type	smallint(6)	No	0	Muestra el tipo de tarea

2.9 Anexo 10 Descripción de la tabla **tasks_departments**

task_departments: Tabla que surge por la relación de muchos a muchos (**n*n**) que se establece entre la tabla **tasks** y la tabla **departments.**

Campo	Tipo	Nulo	Predeterminado	Comentarios
task_id	int(10)	No		Identificador de la tarea
department_id	int(10)	No		Identificador del departamento asociado a la tarea

Anexo 11 Descripción de la tabla **users**

users: Tabla que almacena los datos de los usuarios.

Campo	Tipo	Nulo	Predeterminado	Comentarios
user_id	int(11)	No		Identifica el usuario, asignándole un número como id.
user_contact	int(11)	No	0	Identifica el usuario en la lista de contactos,asignándole un numero como id.
user_username	varchar(255)	No		es el nombre de usuario.

user_password	varchar(32)	No		contiene la contraseña insertada por el usuario.
user_type	tinyint(3)	No	0	Identifica el tipo de usuario, es decir los permisos del mismo.
user_company	int(11)	Si	0	Identifica la empresa a la que pertenece el usuario.
user_department	int(11)	Si	0	Identifica el departamento al que pertenece el usuario.
user_owner	int(11)	No	0	Muestra el id, del usuario que lo creó.

Anexo 12 Descripción de la tabla **users_tasks**

user_tasks: Tabla que surge debido a la relación de (**n*n**)entre la tabla **users** y la tabla **tasks**

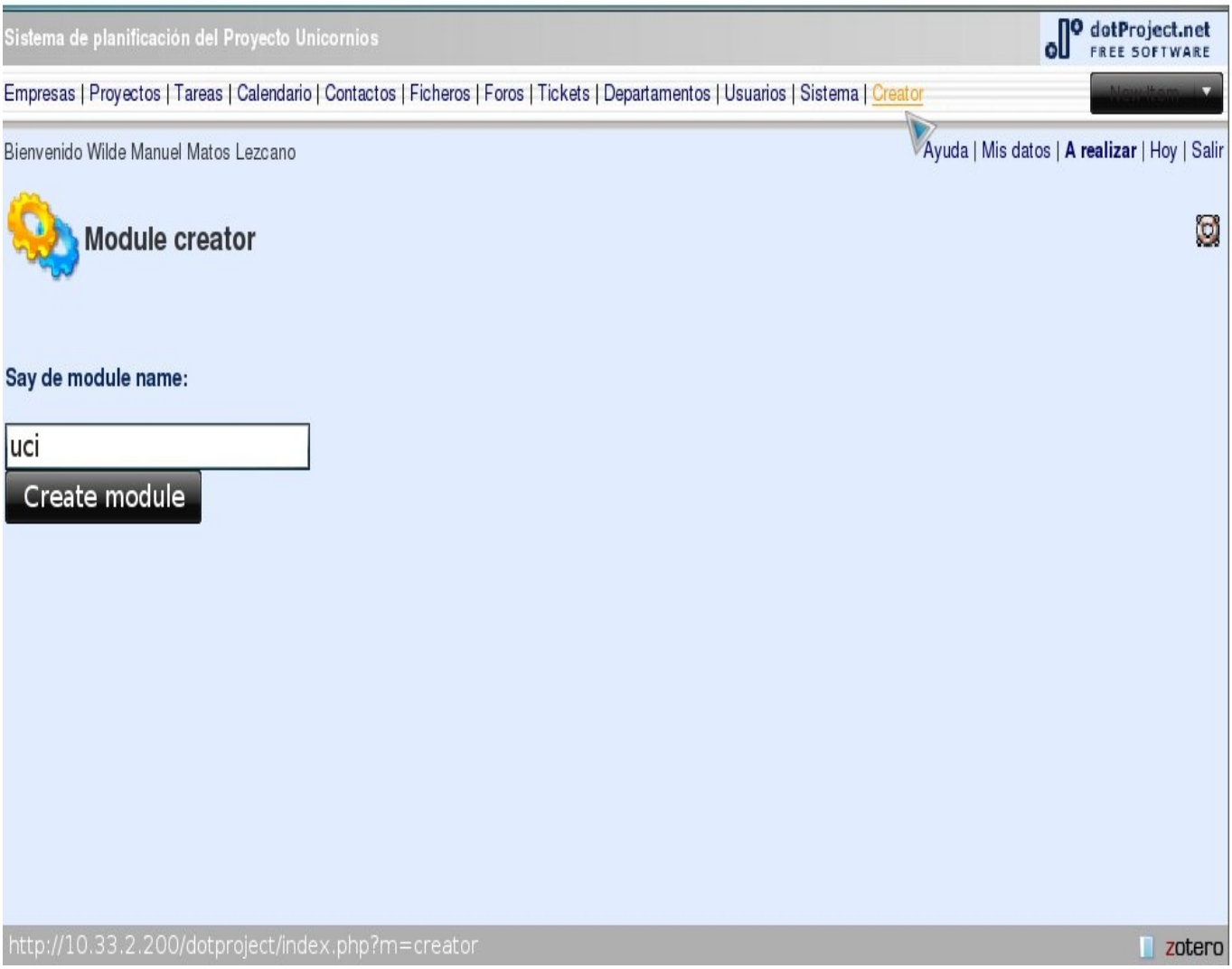
Campo	Tipo	Nulo	Predeterminado	Comentarios
user_id	int(11)	No	0	Identificador del id del usuario
user_type	tinyint(4)	No	0	Muestra el tipo de usuario
task_id	int(11)	No	0	Identificador del id de la tarea
user_task_priority	tinyint(4)	No	0	Muestra el grado de prioridad de esa tarea

Anexo 13 Descripción de la tabla **files**

files: Tabla que almacena los datos de los ficheros

Campo	Tipo	Nulo	Predeterminado	Comentarios
file_id	int(11)	No		Identificador del fichero
file_real_filename	varchar(255)	No		Nombre real del fichero
file_folder	int(11)	No	0	Identificador de la carpeta que contiene el fichero
file_project	int(11)	No	0	Identificador del proyecto al que pertenece el archivo
file_task	int(11)	No	0	Identificador de la tarea al que pertenece el fichero
file_name	varchar(255)	No	0	Nombre del fichero
file_description	text	Si	Null	Descripción del fichero
file_type	varchar(100)	Si	Null	Extensión del fichero
file_size	int(11)	Si	Null	Tamaño del fichero
file_version	float	No	0	Versión del fichero
file_category	int(11)	Si	0	Identificador de la categoría del fichero

Anexo 14 Adicionar nombre del nuevo módulo a crear.



Anexo 15 Configuración de los ficheros **index.php** y **setup.php**