Universidad de las Ciencias Informáticas

FACULTAD 4

Título: Arquitectura de Sistema de la línea Planificación Empresarial y Presupuestada

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

Autor: Guillermo Fernández Elegia

Tutores: Ing.Yoandro Hechavarría Toranzo
Ing. Jessie Guillemi Martín

Ciudad de la Habana, Junio de 2009.

AGRADECIMIENTOS

AGRADECIMIENTOS

A mi madre por todo el sacrificio que ha realizado toda su vida para darme la mejor educación, por todos sus consejos en los buenos y malos momentos, por estar a mi lado, por confiar en mí, por ser una gran madre.

A mi padre por brindarme todo su apoyo, por siempre confiar en mí, por todo el esfuerzo realizado para darme lo mejor.

A mi abuela, a mi madrina y a mi abuelo por estar pendientes de mí, por haberme querido toda la vida, por ayudarme.

A todos mis amigos, que siempre estuvieron conmigo en las buenas y en las malas durante toda la carrera Danny, Joel, Jose Luis, Yankiel, Miguel Angel y todos los demás que son muchos.

A todos los compañeros de mi aula 4503 Betty, Misly, yaneska, Kenner, Yoni, Oilede, y a todos que son muchos.

A mis tutores por brindarme su apoyo durante todo este proceso, en especial a Jessie por haberme ayudado tanto en la realización de este trabajo.

A mi gente del ERP que en muy poco tiempo llegamos a formar una familia Arianna, Sisley, Manuel, Arnoldo, Lidi, Aliuska, Dayana, Drisis, Omarito, Alia, Raúl y todos los demás por su ayuda incondicional en cualquier momento que la necesitara.

A todos muchas gracias.

DEDICATORIA

A mis padres por ayudarme tanto durante toda la carrera y a todos los que confiaron en mí durante mucho tiempo.

RESUMEN

El presente trabajo de diploma recoge un estudio de los elementos de la Arquitectura de Software, durante su desarrollo se realiza el análisis de los mismos con el objetivo de lograr una mejora en el proceso de Planificación Empresarial y Presupuestada de todo el país.

Esta aplicación está basada en un trabajo arquitectónico bien desarrollado, definido con las características fundamentales de las tecnologías y aspectos importantes de la arquitectura y dándole a los miembros del equipo una idea más clara del trabajo que están desarrollando.

El propósito que se persigue es definir la arquitectura de una aplicación Web, la cual debe elevar el nivel de informatización en el proceso de gestión de la Planificación Empresarial y Presupuestada, asegurando que la planificación se realice de forma óptima, rápida y segura.

El desarrollo de esta investigación se sostiene del estudio y la aplicación de metodologías y el uso de herramientas actuales que aseguran que el resultado obtenido sea un producto de software confiable, con un alto grado de calidad.

Palabras clave:

- Arquitectura de Software
- Planificación Empresarial y Presupuestada
- Aplicación Web
- Informatización
- Producto de software

ÍNDICE

ÍNDICE

ÍNDICE DE FIGURAS.

ÍNDICE DE TABLAS.

INTRODUCCIÓN

Las naciones del mundo están llamadas a alcanzar la economía del conocimiento y el país no se encuentra ajeno a esta actividad. Hoy se lucha por perfeccionar la economía cubana mientras se lleva a cabo la recuperación paulatina de todos los sectores. Sin duda alguna las entidades recibirían con gran beneplácito un software que gestione sus recursos de forma integral, un sistema de gestión que esté diseñado para modelar e informatizar la mayoría de los procesos en las empresas (las áreas de finanzas, comercial, logística, planificación). Estos sistemas son extremadamente costosos, y una vez que se implantan con éxito traen una serie de beneficios importantes.

Las ventajas que aporta el uso de estos sistemas como herramienta de planificación y control son numerosas. Sin embargo lo más destacable es que unifica y ordena toda la información de la entidad en un sólo lugar, de este modo cualquier suceso queda a la vista de forma inmediata, posibilitando la toma de decisiones de forma más rápida y segura, acortando los ciclos productivos. Esto permite mantener un control e incrementar la calidad de los servicios y productos. La información fluye por toda la empresa eliminando la improvisación por falta de datos.

En el mundo existen varias soluciones de este tipo de software como son: mySAP ERP, SAGE MAS 500; y también nacionales como el Versat-Sarasola, pero muchas de ellas no son capaces de integrar todas las funcionalidades que se requieren en una entidad y otras resultan demasiado costosas para usarlas.

Con la planificación, con la cual se definen los objetivos o metas de la organización, estableciendo una estrategia general para alcanzar esas metas y desarrollar una jerarquía completa de planes para coordinar las actividades .

- Se ocupa tanto de los fines o metas (¿qué hay que hacer?) como de los medios o herramientas (¿cómo debe hacerse?).

- La planificación define una dirección, se reduce el impacto del cambio, se minimiza el desperdicio y se establecen los criterios utilizados para controlar.

- Da dirección a los gerentes y a toda la organización. Cuando los empleados saben a donde va la organización y en que deben contribuir para alcanzar ese objetivo, pueden coordinar sus actividades, cooperar entre ellos y trabajar en equipos.

INTRODUCCIÓN

- Sin la planificación, los departamentos podrían estar trabajando con propósitos encontrados e impedir que la organización se mueva hacia sus objetivos de manera eficiente.

- Establece objetivos o estándares que se usan para controlar (1) .

En estos momentos, el país presenta deficiencias en los procesos de planificación, se hacen gastos excesivos en millones de dólares en sistemas propietarios lo cual afecta nuestra economía; no se tiene un control centralizado en los procesos que se llevan a cabo; existe una gran fragmentación de datos en los diferentes niveles; se incurren en demoras en la entrega de documentos; existen errores humanos en cálculos a la hora de planificar; gastos excesivos de material de oficina; ineficiencia en el trabajo de planificación en las diferentes empresas; el hecho de no contar con una aplicación para modelar todos los procesos de la planificación en el país hace aún más difícil la tarea para los profesionales, pues esta puede llegar a ser muy dinámica.

Ante esta situación problemática, la dirección del país propuso la realización de un sistema de gestión para informatizar los procesos que se realizan en las entidades surgiendo en el 2008 el proyecto SIGE (sistema integral de gestión de entidades) dentro de la Universidad de las Ciencias Informáticas (UCI). El mismo está estructurado por líneas de producción, siendo una de ellas la de Planificación Empresarial y Presupuestada. Este trabajo estará enmarcado en la arquitectura de sistema que requiere dicha línea por lo que el **problema** a resolver sería:

¿Cómo definir una arquitectura para el Subsistema de Planificación Empresarial y Presupuestada del sistema Cedrux, que garantice la gestión de la planificación en las entidades cubanas?

El **objeto de estudio** será la arquitectura de software y tendrá como **campo de acción** la arquitectura de software orientada a componentes.

Para solucionar el problema planteado se propone como **objetivo general:**

Definir una arquitectura de software apropiada y generar los artefactos necesarios para garantizar la calidad y eficiencia requerida en el Subsistema de Planificación Empresarial y Presupuestada.

Los **objetivos específicos** que se desean alcanzar son:

- Realizar un estudio del estado del arte teniendo en cuenta la evolución de la arquitectura de software en la actualidad.

- Definir la arquitectura de sistema del Subsistema Planificación Empresarial y Presupuestada.
- Evaluar la arquitectura definida.

Las **tareas** que se realizarán para dar solución al problema y cumplir con los objetivos trazados son:

1. Investigar y documentar la evolución de la arquitectura de software en la actualidad.
2. Definir la arquitectura del subsistema de acuerdo a la metodología y herramientas propuestas por el proyecto.
3. Definir y ejemplificar los patrones de diseño para emplearlos en el desarrollo del subsistema de Planificación Empresarial y Presupuestada.
4. Definir los componentes de la línea así como la integración existente entre ellos tanto interna como externa.
5. Priorizar los componentes definidos.
6. Proponer la arquitectura de software para el subsistema.
7. Analizar los resultados obtenidos y la calidad de la arquitectura del subsistema desarrollado.

Los **resultados esperados** de este trabajo son:

- Obtener la documentación de la investigación
- Obtener la arquitectura del subsistema Planificación Empresarial y Presupuestada.

El presente trabajo estará conformado por 3 capítulos de los cuales se brinda información acerca de ellos.

En el capítulo 1 se encontrará la fundamentación teórica dentro de la cual habrá un estudio referencial de los estilos arquitectónicos, patrones y las herramientas para el desarrollo de la arquitectura de sistema.

En el capítulo 2 se realizará la propuesta arquitectónica desarrollando los distintos artefactos para el desarrollo de la misma.

El capítulo 3 muestra un análisis de los resultados obtenidos.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción.

En el presente capítulo se realizará un estudio del marco teórico sobre los estilos y patrones arquitectónicos y de diseño para desarrollar aplicaciones web partiendo de los principales conceptos para lograr su entendimiento, además de mencionar las herramientas utilizadas para el desarrollo de este trabajo.

1.2 ¿Qué es la Arquitectura del Software?

En los inicios de la informática, la programación se consideraba un arte, debido a la dificultad que entrañaba para la mayoría de las personas, con el paso del tiempo se han ido desarrollando metodologías para conseguir esos propósitos (2). Pero no es hasta 1968, cuando Edsger Dijkstra[1], de la Universidad Tecnológica de Eindhoven en Holanda propuso que se estableciera una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera y, aunque no utiliza el término arquitectura para describir el diseño conceptual del software, sus conceptos sientan las bases para lo que luego evolucionaría como lo que hoy se conoce como *Arquitectura del Software (AS).* Una definición reconocida es la de Clements: “La AS es, a grandes rasgos, una vista del sistema que incluye los principales componentes del mismo, la conducta de esos componentes según se 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. Garlan and Shaw en 1994 y 1996 la conceptualizaron como una colección de componentes y conectores unidos con una descripción de la interacción entre componentes y conectores (3). La definición oficial de Arquitectura del Software es la que da la IEEE Std 1471-2000 que dice así: “La Arquitectura del Software es la organización fundamental de un sistema

[1] Dijkstra estudió física teórica en la Universidad de Leiden. Entre sus contribuciones a la informática está el algoritmo de caminos mínimos; también conocido como Algoritmo de Dijkstra. Recibió el Premio Turing en 1972.Era conocido por su baja opinión de la sentencia GOTO en programación, que culminó en 1968 con el artículo *Go To Statement Considered Harmful*, visto como un paso importante hacia el rechazo de la expresión GOTO y de su eficaz reemplazo por estructuras de control tales como el bucle while.

formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución". Puede verse que todas estas
definiciones se basan o se enfocan en como interactúan los componentes y conectores.

Componente: Es una organización no-trivial, casi independiente, y parte reemplazable de un sistema que cumple una clara función en el contexto de una arquitectura bien definida. Un componente se ajusta y proporciona la realización física de un conjunto de interfaces (3). Se entiende por *componentes* los bloques de construcción que conforman las partes de un sistema de software. A nivel de lenguajes de programación, pueden ser representados como módulos, clases, objetos o un conjunto de funciones relacionadas (4).

Conectores: Definen la interacción entre los componentes y describen las reglas que gobiernan esas interacciones (3).

Interfaces: Un componente define una interfaz a través de un conector que une vínculos con otro componente. Un componente puede tener múltiples interfaces (3). Desde Dijkstra hasta la actualidad existen muchas definiciones de arquitectura del software, de hecho, existen grandes compilaciones de definiciones alternativas o contrapuestas, como la colección que se encuentra en el SEI (Software Engineering Institute) (4), y no parece que ninguna de ellas haya sido totalmente aceptada.

En un sentido amplio se puede estar de acuerdo en que la Arquitectura del Software es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación y tiene la responsabilidad de:

- Definir los componentes (módulos y subsistemas) principales.
- Definir las responsabilidades que tendrá cada uno de ellos.
- Definir la interacción que existirá entre dichos componentes.
- Control y flujo de datos
- Secuenciación de la información
- Protocolos de interacción y comunicación
- Ubicación en el hardware

Independientemente de las diversas definiciones, *¿Qué proporciona?* La arquitectura de software establece los fundamentos para que analistas, diseñadores, programadores, trabajen en una línea común que permita alcanzar los objetivos y necesidades del sistema de información, permitiendo la organización en todo el desarrollo, encarnada en sus componentes, las relaciones entre ellos y con su entorno, y los principios que gobiernan su diseño y evolución, define la organización fundamental de un sistema. La arquitectura de software está afectada no solo por la estructura y el comportamiento sino también por el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de comprensión, las restricciones y compromisos económicos y tecnológicos y la estética (5).

1.3. Estilos Arquitectónicos.

Las soluciones de diseño arquitectónicas que son comunes y reusables a lo largo de años de experiencia se han ido agrupando en lo que más tarde se les llamó *estilos*. El éxito del diseño de la arquitectura de software depende de los estilos que se decida utilizar para el desarrollo de la misma. 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 re-utilizarlos en situaciones semejantes que se presenten en el futuro (3). Los estilos arquitectónicos son arquitecturas comunes, marcos de referencia arquitectónica prototípica, formas comunes, clases de sistemas.

Un estilo arquitectónico define tanto un vocabulario de tipos de componentes y conectores (6), como un conjunto de restricciones sobre cómo combinar esos componentes y conectores, que sirven para sintetizar estructuras de soluciones que luego serán refinadas a través del diseño. Definir patrones posibles de las aplicaciones, evitando errores arquitectónicos, permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos. Existe una gran variedad de estilos arquitectónicos las cuales tienen diversas clasificaciones entre las que se encuentran:

- Estilos de flujo de datos
- Estilos centrados en datos
- Estilos de llamada y retorno

- Estilos de código móvil
- Estilos heterogéneos

A diferencia de los patrones de diseño, que son centenares, los estilos se ordenan en seis o siete clases fundamentales y unos veinte ejemplares, como máximo. Las arquitecturas complejas o compuestas resultan del agregado o la composición de estilos más básicos. Existen pocos estilos que encapsulan una enorme variedad de configuraciones.

1.3.1. Arquitectura basada en Componentes.

Los sistemas de software basados en componentes se basan en principios definidos por una ingeniería de software específica. El objetivo de la tecnología de componentes software es construir aplicaciones complejas mediante ensamblado de los mismos que han sido previamente diseñados por otras personas a fin de ser rehusados en múltiples aplicaciones. La arquitectura de software de una aplicación basada en componentes consiste en uno o un número pequeño de componentes específicos de la aplicación (que se diseñan específicamente para ella), que hacen uso de otros muchos componentes prefabricados que se ensamblan entre sí para proporcionar los servicios que se necesitan en la aplicación. En la tecnología de componentes la interfaz constituye el elemento básico de interconectividad. Cada componente debe describir de forma completa las interfaces que ofrece, así como las interfaces que requiere para su operación. Y debe operar correctamente con independencia de los mecanismos internos que utilice para soportar la funcionalidad de la interfaz.

Características muy relevantes de la tecnología de programación basada en componentes son la modularidad, la reusabilidad y compatibilidad y en todos ellos coincide con la tecnología orientada a objetos de la que se puede considerar una evolución. Sin embargo, en la tecnología basada en componentes también se requiere robustez ya que los componentes han de operar en entornos mucho más heterogéneos y diversos.

El desarrollo de software basado componentes es la evolución natural de la ingeniería software para mejorar la calidad, disminuir los tiempos de desarrollo y gestionar la creciente complejidad de los sistemas.

Entre las ventajas que posee este estilo arquitectónico se puede ver que: cuando se tienen nuevos requerimientos a implementar, se pueden proveer nuevos componentes, sin tocar los ya existentes, no afectándolos así por los nuevos requerimientos. Esos factores permiten a la programación orientada a componentes reducir el costo a lo largo de la etapa de mantenimiento, esto es un factor esencial en

la mayoría de los negocios, en los cuales sé esta extendiendo el uso de la tecnología de componentes (7).

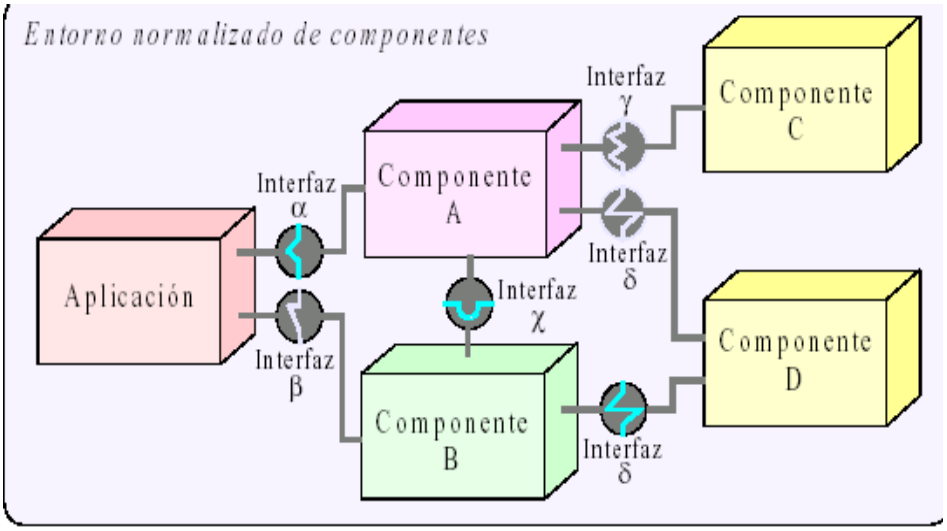


Figura 1: Arquitectura basada en componentes

1.3.2. Arquitectura Modelo Vista Controlador (MVC)

Patrón clásico del diseño Web conocido como arquitectura MVC, que está formado por tres niveles:

- El modelo representa la información con la que trabaja la aplicación, es decir, su lógica de negocio.

- La vista transforma el modelo en una página Web que permite al usuario interactuar con ella.

- El controlador se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista.

La arquitectura 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. Por ejemplo, si una misma aplicación debe ejecutarse tanto en un navegador estándar como en 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 la vista de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista

y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación.

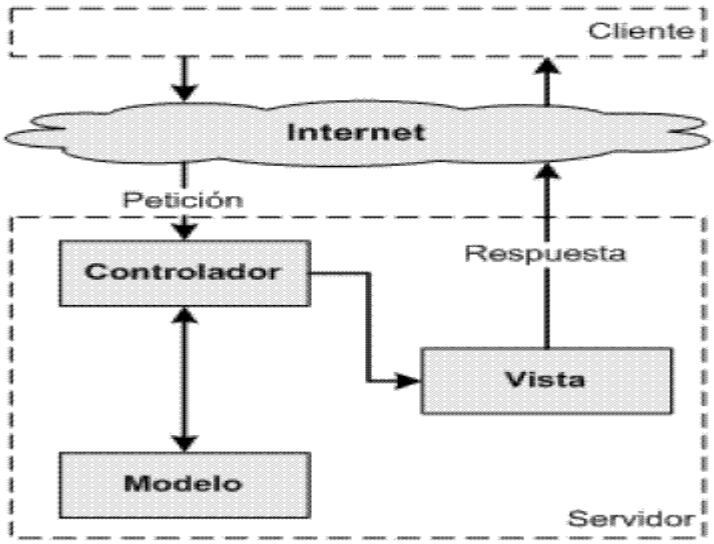


Figura 2: El patrón MVC

Este modelo de arquitectura presenta varias ventajas:

- Hay una clara separación entre los componentes de un programa; lo cual permite implementarlos por separado

- La conexión entre el Modelo y sus Vistas es dinámica; se produce en tiempo de ejecución, no en tiempo de compilación.

- Al incorporar el modelo de arquitectura MVC a un diseño, las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Posteriormente, si uno de los Componentes, se observa que funciona mal puede reemplazarse sin que las otras piezas se vean afectadas.
- La porción del programa que transforma los datos dentro del Modelo en una presentación gráfica es la vista. La vista incorpora la visión del Modelo a la escena; es la representación gráfica de la escena desde un punto de vista determinado, bajo condiciones de iluminación determinadas. El Controlador sabe que puede hacer el modelo e implementa la interfaz de usuario que permite iniciar la acción (8).

1.4. Patrones Arquitectónicos.

Son patrones de alto nivel que fijan la arquitectura global de una aplicación, “los patrones se refieren más bien a prácticas de re-utilización y los estilos conciernen a teorías sobre la estructuras de los sistemas a veces más formales que concretas” (9). Un patrón es una regla que consta de tres partes, la cual expresa una relación entre un contexto, un problema y una solución. Estas son*:*

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. Son plantillas para arquitecturas de software concretas, que especifican las propiedades estructurales de una aplicación y tienen un impacto en la arquitectura de subsistemas. La selección de un patrón arquitectónico es, por tanto, una decisión fundamental de diseño en el desarrollo de un sistema de software.

Layers (Capas): Consiste en estructurar aplicaciones que pueden ser compuestas en grupos de subtareas las cuales se clasifican de acuerdo a un nivel particular de abstracción.

Pipes and Filters (Tuberías y Filtros): Provee una estructura para los sistemas que procesan un flujo de datos. Cada paso de procesamiento está encapsulado en un componente filtro (filter). El dato pasa a través de conexiones (pipes), entre filtros adyacentes.

Blackboard (Pizarra): Aplica para problemas cuya solución utiliza estrategias no determinísticas. Varios subsistemas ensamblan su conocimiento para construir una posible solución parcial ó aproximada.

Broker (corredor): Puede ser usado para estructurar sistemas de software distribuido con componentes desacoplados que interactúan por invocaciones a servicios remotos. Un componente broker es responsable de coordinar la comunicación, como el reenvío de solicitudes, así como también la transmisión de resultados y excepciones.

Model-View-Controller (Modelo Vista Controlador): Divide una aplicación interactiva en tres componentes. El modelo (model) contiene la información central y los datos. Las vistas (view) despliegan información al usuario. Los controladores (controllers) capturan la entrada del usuario. Las vistas y los controladores constituyen la interfaz del usuario.

Presentation-Abstraction-Control (Presentación Abstracción Control): Define una estructura para sistemas de software interactivos de agentes de cooperación

organizados de forma jerárquica. Cada agente es responsable de un aspecto específico de la funcionalidad de la aplicación y consiste en tres componentes: presentación, abstracción y control.

Microkernel: Aplica para sistemas de software que deben estar en capacidad de adaptar los requerimientos de cambio del sistema. Separa un núcleo funcional mínimo del resto de la funcionalidad y de partes específicas pertenecientes al cliente.

Reflection (Reflexión): Provee un mecanismo para sistemas cuya estructura y comportamiento cambia dinámicamente. Soporta la modificación de aspectos fundamentales como estructuras tipo y mecanismos de llamadas a funciones.

1.5. Patrones de Asignación de responsabilidades.

Los patrones de diseño surgen ante la necesidad de la reutilización de diseños y de soluciones a problemas frecuentes encontrados por los ingenieros. Con la reutilización se consigue la reducción de tiempos y la disminución del esfuerzo de mantenimiento, esto trae consigo mayor eficiencia y consistencia en el diseño de la solución.

Graig Larman en su libro UML y Patrones (10) menciona un grupo de patrones relacionados con el diseño de software, los cuales son llamados patrones GRASP (General Responsibility Asignment Software Patterns) y describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones, algunos de ellos tienen mucha relación con los problemas básicos del diseño y deben su nombre a estos. La asignación correcta de las responsabilidades en el diseño orientado a objetos garantiza la alta cohesión de las clases y el bajo acoplamiento de los mismos, lo que posibilita más extensibilidad, adaptabilidad y menos tiempo para el mantenimiento del diseño, por ello estos patrones son parejas de problema/solución con un nombre, que codifican buenos principios y sugerencias relacionados frecuentemente con la asignación de responsabilidades. Estos patrones pueden ser aplicados preferiblemente durante la preparación de un diagrama de interacción o colaboración ya que es en esta actividad del diseño donde el diseñador estructura las responsabilidades y la colaboración entre los componentes de diseño (clases).

Los patrones GRASP son los siguientes:

- Experto
- Creador
- Alta Cohesión
- Bajo Acoplamiento
- Controlador

El patrón ***Experto*** trata de resolver el problema de como asignan las responsabilidades en el diseño orientado a objetos. Un modelo de clase puede definir docenas y hasta cientos de clases de software, y una aplicación tal vez requiera el cumplimiento de cientos o miles de responsabilidades. Para dar solución a esta problemática el patrón experto plantea asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad (10).

El patrón ***Creador*** trata de resolver el problema de diseño de quién debería ser responsable de crear una nueva instancia de alguna clase. La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. En consecuencia, conviene contar con un principio general para asignar las responsabilidades concernientes a ella. El diseño, bien asignado, puede soportar un bajo acoplamiento, una mayor claridad, el encapsulamiento y la reutilizabilidad.

Para dar solución a esta problemática el patrón creador plantea asignarle a la clase B la responsabilidad de crear una instancia de clase A en uno de los siguientes casos:

- B agrega los objetos A.
- B contiene los objetos A.
- B registra las instancias de los objetos A.
- B utiliza específicamente los objetos A.
- B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado (así que B es un Experto respecto a la creación de A).
- B es un creador de los objetos A.

Si existe más de una opción, prefiera la clase B que agregue o contenga la clase A.

El patrón ***Bajo Acoplamiento*** trata de resolver el problema de diseño de cómo dar soporte a una dependencia escasa y a un aumento de la reutilización.

Para dar solución a esta problemática el patrón bajo acoplamiento plantea asignar una responsabilidad a cada clase o grupo de clases según su contexto que no involucre recursos concurrentes de otras clases que no están enmarcadas en este contexto de modelación del problema, para mantener de este modo bajo el acoplamiento en el diseño (10).

El patrón ***Alta Cohesión*** trata de resolver el problema de diseño de cómo mantener la complejidad dentro de límite manejable.

Para dar solución a esta problemática el patrón alta cohesión plantea asignar una responsabilidad a cada clase o grupo de clases según su contexto que no asuma responsabilidades fuera de su dominio de modelación, si una clase esta modelando la gestión de personas, no debe recibir la responsabilidad de autenticar un usuario, esa función le debe corresponder a otra clase, de esta forma se puede lograr una alta

cohesión en el diseño del sistema. La alta cohesión al igual que el bajo acoplamiento facilita una diseño mantenible, fácil de reutilizar y adaptar, más legibilidad para los programadores y diseñadores en general; la extensibilidad y flexibilidad del diseño aumentan, ya que las clases usan sólo lo que necesitan y en su mayoría son recursos propios, por otro lado, cada cual hace lo que le corresponde hacer según la parte del problema que está modelando (10).

El patrón ***Controlador*** trata de resolver el problema de diseño de quién debería encargarse de atender un evento del sistema. Los eventos del sistema pueden ser varios en un mismo formulario, y cada uno demanda de una validación de precondiciones definidas en la especificación de los requerimientos, en ocasiones suele ser un problema atenderlos a todos de forma consistente, y que ante la repetición de alguno de ellos dentro del mismo formulario no exista la necesidad de reprogramar el mismo.

Un Controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Define además el método de su operación.

Para dar solución a esta problemática el patrón Controlador plantea asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones:

- El "sistema" global (controlador de fachada).
- La empresa u organización global (controlador de fachada).
- Algo en el mundo real que es activo (por ejemplo, el papel de una persona) y que pueda participar en la tarea (controlador de tareas).
- Un manejador artificial de todos los eventos del sistema de un caso de uso, generalmente denominados **"Manejador<NombreCasodeUso>"** (controlador de casos de uso). Se utiliza la misma clase de controlador con todos los eventos del sistema en el mismo caso de uso.

Los patrones GRASP sin dudas son la base de inspiración de otros patrones más poderosos y que han ganado muchísima reputación en el estado del arte del diseño de software actual, además de consolidar una norma de consulta perenne para desarrollar un diseño eficiente, flexible, adaptable, reutilizable y eficaz.

1.6. Patrones de diseño.

En el diseño del software también surgen otro gran número de problemas como son los relacionados con el uso de memoria, la creación de trazas de operaciones, la creación de familias de objetos, o incluso la gestión y configuración de entornos entre otros que han constituido el punto de partida para muchos diseñadores, teniéndose en la actualidad una gran gama de patrones a utilizar con estos fines, por el grado de encapsulamiento que da la solución al problema, por la limpieza y facilidad de entendimiento, además de la calidad y del aporte creativo al diseño, se incluyen en este trabajo el grupo de patrones conocidos como Patrones “GoF”.

Los autores del trabajo “Design Patterns: Elements of reusable object-oriented software” (11), proponen un número grande de patrones agrupados en tres categorías, que resuelven casi todas las problemáticas básicas del diseño de software, es muy difícil ver hoy un diseño de software que no haya reutilizado o se haya inspirado una de las propuestas de solución a problemas que se plantee en estos patrones.

A continuación se hace referencia a las categorías en la que se agrupan estos patrones, si se desea profundizar cada uno de estos patrones ver (11).

Estos tipos de patrones se clasifican en 3 estilos diferentes según su utilización: creación, estructurales, comportamiento.

- Creación**:** Es la encargada de crear objetos cuando sus creaciones requieren tomar decisiones.
- Estructurales**:** Describen la forma en que diferentes tipos de objetos pueden ser organizados para trabajar unos con otros.
- Comportamiento**:** Organiza, maneja y combina comportamientos.

Cada tipo de patrón de estos presenta una gama de patrones, que los encapsula, cumpliendo con las características de los mismos.

Tabla 1: Patrones de diseño

Creación	Estructurales	Comportamiento

Abstract Factory	Adapter	Chain of Responsability
Builder	Bridge	Command
Factory Method	Composite	Interpreter
Prototype	Decorator	Iterator
Singleton	Facade	Mediator
	Flyweight	Memento
	Proxy	Observer
		State
		Strategy
		Template Method
		Visitor

Los patrones son una gran familia que permiten el ahorro del tiempo, así como ser exactos a la hora de darle respuesta a un problema determinado, si cada patrón se emplea en el contexto que él requiere o sea para lo cual fue destinado, la eficiencia y el resultado a obtener será elevado. En la actualidad, cada vez se hacen más necesario el uso de estos patrones, por la complejidad que va obteniendo el software y las necesidades del hombre, que cada vez necesita que el software sea más eficiente.

1.7. Estilos y Patrones Arquitectónicos.

Se estima que los estilos se pueden comprender como clases de patrones, o tal vez más adecuadamente como lenguajes de patrones. Un estilo proporciona de este modo un lenguaje de diseño con un vocabulario y un framework a partir de los cuales los arquitectos pueden construir patrones de diseño para resolver problemas específicos.

Algunos patrones coinciden con los estilos hasta en el nombre con que se los designa (12).

Tabla 2: Estilos y patrones

Estilo Arquitectónico	**Patrón Arquitectónico**
Sólo describe el esqueleto estructural y general para aplicaciones	Existen en varios rangos de escala comenzando con patrones que definen la estructura básica de una aplicación
Son independientes del contexto al que puedan ser aplicados.	Partiendo de la definición de patrón, requieren de la especificación de un contexto del problema
Cada estilo es independiente de los otros	Depende de patrones más pequeños que contiene, patrones con los que interactúa, o de patrones que lo contengan
Expresan técnicas de diseño desde una perspectiva que es independiente de la situación actual de diseño.	Expresa un problema recurrente de diseño muy específico, y presenta una solución para él, desde el punto de vista del contexto en el que se presenta
Son una categorización de sistemas.	Son soluciones generales a problemas comunes

Como se ha podido ver, la diferencia entre los estilos y patrones arquitectónicos está en el nivel de abstracción en el que uno se mueve. Los estilos se aplican a un nivel muy alto de abstracción, en el cual no interesa saber cuál es la semántica de los elementos que se están utilizando, sólo se habla de filtros, tubos, u objetos sin interesarnos por lo que están representando. En el caso de los patrones, tenemos en cuenta el significado de los filtros, tubos, objetos, tal como hemos visto en los patrones de descomposición en capas, en el que se habla de presentación, dominio de aplicación y de datos o en el de Módulo Tabla, que diferencia entre un objeto que representa un elemento, una línea de una tabla o registro u otro objeto que representa la totalidad de la tabla (6).

Podría decirse que mientras los estilos han enfatizado descriptivamente las configuraciones de una arquitectura, desarrollando incluso lenguajes y notaciones capaces de expresarlas formalmente, los patrones, aún los que se han caracterizado

como arquitectónicos, se encuentran más ligados al uso y más cerca del plano físico, sin disponer todavía de un lenguaje de especificación. Los patrones se refieren más bien a prácticas de reutilización y los estilos conciernen a teorías sobre la estructuras de los sistemas a veces más formales que concretas (13).

De forma que, aunque en ocasiones para referirse a ellos se mezclan patrones con estilos y viceversa, se debe pensar que los estilos son una forma mucho más amplia o global de definir una arquitectura, desde un nivel de abstracción más alto, sin muchos detalles y los patrones van más ligados a la implementación de estos estilos, un poco más abajo en cuanto a la abstracción, directamente con el código o el lenguaje de programación que se utiliza.

1.8. Ambiente de desarrollo (Development Environment).

El ambiente de desarrollo es algo imprescindible en la producción de software. Es donde se definen el conjunto de herramientas y tecnologías (framework o marco de trabajo), versiones a usar y su integración, que intervienen en un proceso de desarrollo de software (13). A continuación se presentan un conjunto de herramientas utilizadas en el desarrollo de este trabajo las cuales fueron definidas por la línea de arquitectura del proyecto Cedrux como son: un ambiente de desarrollo integrado (IDE), un servidor web, un servidor para control de versiones, un sistema gestor de base de datos, herramientas de modelado, los frameworks utilizados, herramientas de diseño web. Se exponen sus características y ventajas para poder tener un mayor conocimiento de sus prestaciones y valorar su utilidad.

1.9. Ambiente de desarrollo integrado (Integrated Development Environment, IDE).

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

1.9.1. Zend Studio for Eclipse.

Zend Studio para Eclipse IDE es de la próxima generación en la familia Zend Studio. Es la última versión del popular entorno de programación integrado. Diseñado para desarrolladores profesionales de PHP, esta nueva versión combina un IDE versátil y

potente con las capacidades de expansión del ecosistema del proyecto Eclipse. Dispone de un entorno mucho más flexible y profesional para controlar todo el ciclo de vida de un desarrollo. Entre sus funcionalidades, destacaría las capacidades de refactorización[2] del código fuente, funcionalidad que permite adecuar el comportamiento externo de una función/clase sin cambiar el funcionamiento interno, que junto a los nuevos asistentes, programas que mediante pantallas interactivas facilitan y sirven de guía para el uso de otros programas o aplicaciones, y capacidades de generación de código facilitarán el trabajo a los desarrolladores.

Características nuevas que incluye:

- Generación de código.
- Código de Cobertura.
- El acceso al ecosistema de pluggins de Eclipse.
- Apoyar el desarrollo de múltiples idiomas.
- Mejora el Editor de PHP con el formato avanzado, para listas de tareas y problemas de vista.
- Mejora del soporte de JavaScript.
- Mejora de apoyo, incluyendo HTML, HTML WYSIWYG, Código plegables, arrastrar y soltar los componentes y más.
- Mejora de depuración y Perfiles con Path Mapping.
- Mejora de Zend Framework con el apoyo del Proyecto Framework, plantillas y código MVC.

1.10. Framework (Marco de trabajo).

En el desarrollo de software, un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje de scripting entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo, la cual extiende o utiliza las aplicaciones del dominio. Estos son diseñados con el intento de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional. Es el esqueleto sobre

[2] La refactorización es el proceso de tomar código que funciona y hacer que funcione mejor. Normalmente "mejor" significa "más rápido", aunque puede significar también que "usa menos memoria" o "usa menos espacio en disco" o simplemente "es más elegante". Independientemente de lo

el cual varios objetos son integrados para una solución dada. También se puede definir en el contexto de la programación como un set de funciones o código genérico que realiza tareas comunes y frecuentes en todo tipo de aplicaciones (creación de objetos, conexión a base de datos, limpieza del código). Esto brinda una base sólida sobre la cual desarrollar aplicaciones concretas y permite obviar los componentes más triviales y genéricos del desarrollo.

En general, son construidos en base a lenguajes orientados a objetos. Esto permite una mejor modularización de los componentes y óptima reutilización de código. Además, en la mayoría de los casos un framework implementará uno o más patrones de diseño de software que aseguren la escalabilidad del producto (14).

1.10.1. ExtJS.

Es un framework para JavaScript utilizado en el desarrollo de aplicaciones Web con AJAX.[3] Tiene una librería inmensa que permite configurar las interfaces Web de manera semejante a aplicaciones de escritorio.

Tiene incluidos la mayoría de los controles de los formularios Web incluyendo Grids para mostrar datos y elementos semejantes a la programación de escritorio como los formularios, paneles, barras de herramienta, menús y muchos otros. Dentro de su librería de componentes incluye componentes para el manejo de datos, lectura de XML, lectura de datos JSON[4] e implementaciones basadas en AJAX.

1.10.2. Zend Framework.

Los frameworks por lo general presentan una estructura organizada que obliga a los programadores a seguir estándares y a trabajar de manera organizada. El uso de estas aplicaciones ha demostrado que la organización de la programación influye notablemente en la calidad de las aplicaciones.

Zend Framework es uno de los más utilizados para PHP y utiliza el estilo MVC como base de su funcionamiento. Es fácilmente integrable a las aplicaciones debido a su

---

que signifique para usted, su proyecto, en su entorno, la refactorización es importante para la salud a largo plazo de cualquier programa.

[3] **AJAX**, acrónimo de ***A**synchronous **J**avaScript **A**nd **X**ML* (Javascript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

[4] **JSON:** acrónimo de "**J**ava**S**cript **O**bject **N**otation", es un formato ligero para el intercambio de datos, es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

composición ya que contiene diferentes clases de gran utilidad como por ejemplo en la búsqueda dinámica de ficheros a incluir.

Cuenta con un importante mecanismo de manejo de controladores y vistas por lo que se propone tenerlo en cuenta para el diseño de estos dos componentes de la arquitectura.

1.10.3. Doctrine PHP.

Doctrine es un potente y completo sistema ORM (Object Relational Mapper, mapeador de objetos relacionales) para PHP 5.2+ con un DBAL (Data Base Abstraction Layer, capa de abstracción Base de Datos) incorporado.

Entre muchas otras cosas da la posibilidad de exportar una base de datos existente a sus clases correspondientes y también a la inversa, es decir convertir clases (convenientemente creadas siguiendo las pautas del ORM) a tablas de una base de datos.

Su principal ventaja radica en poder acceder a la base de datos utilizando la programación orientada a objetos (POO) debido a que doctrine utiliza el patrón Active Record (Registro Activo) para manejar la base de datos, tiene su propio lenguaje de consultas y trabaja de manera rápida y eficiente. Es fácilmente integrado a los principales frameworks de desarrollo utilizados actualmente.

1.11. Lenguajes de Modelado.

1.11.1. Lenguaje unificado de modelado (UML).

UML es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Es importante resaltar que UML es un lenguaje para especificar y no para describir métodos o procesos. Se utiliza para definir y detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software pero no especifica en sí mismo qué metodología o proceso usar.

1.12. Herramienta de modelado.

1.12.1. Visual Paradigm.

Visual Paradigm para UML es una herramienta CASE que utiliza UML como lenguaje de modelado, soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones con calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Visual Paradigm ofrece:

- Entorno de creación de diagramas para UML.
- Diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad de integrarse en los principales IDEs.
- Disponibilidad en múltiples plataformas.

1.13. Servidor Web.

Un servidor web es un programa que implementa el *protocolo HTTP* (*hypertext transfer protocol, protocolo de traslado de hipertexto*). Este protocolo está diseñado para transferir lo que se llama hipertextos, páginas web o páginas HTML (hypertext markup language, lenguaje marcado de hipertexto): textos complejos con enlaces, figuras, formularios, botones y objetos incrustados como animaciones o reproductores de música.

Es un programa que se ejecuta continuamente en un ordenador (también se emplea el término para referirse al ordenador que lo ejecuta), manteniéndose a la espera de peticiones por parte de un cliente (un navegador web) y que responde a estas adecuadamente, mediante una página web que se exhibirá en el navegador o mostrando el respectivo mensaje si se detectó algún error.

1.13.1. Servidor Web Apache.

El servidor HTTP Apache, de código abierto para plataformas Unix (BSD, GNU/Linux), Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. Cuando comenzó su desarrollo en 1995 se basó inicialmente en el código del popular NCSA HTTPd 1.3, pero más tarde fue reescrito por completo. Apache presenta entre otras características mensajes de error altamente configurables, bases de datos de autenticación y negociado de contenido.

Ventajas:

- Modular.
- Open Source.
- Multi-plataforma.
- Extensible.
- Popular (fácil conseguir ayuda/soporte).
- Gratuito.

1.14. SGBD.

Un Sistema Gestor de Bases de Datos (Data Base Management System DBMS) consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a esos datos. El objetivo primordial de un SGBD es proporcionar un entorno que sea a la vez conveniente y eficiente para ser utilizado al extraer y almacenar información de la base de datos. Es una aplicación que permite a los usuarios definir, crear y mantener la base de datos, y proporciona acceso controlado a la misma (15).

1.14.1. PostgreSQL.

PostgreSQL es un Sistema de Gestión de Bases de Datos Objeto-Relacionales (ORDBMS), de código abierto desde su inicio y con una activa y dedicada comunidad de desarrolladores. Ofrece una potencia adicional sustancial al incorporar los siguientes cuatro conceptos adicionales básicos en una vía en la que los usuarios pueden extender fácilmente el sistema: clases, herencia, tipos, funciones. Otras características aportan potencia y flexibilidad adicional.

1.15. Lenguajes de Programación.

1.15.1. Lenguaje de programación PHP.

Lenguaje utilizado del lado del servidor. Es de propósito general dedicado al desarrollo de páginas web dinámicas que acceden a bases de datos. Desde su aparición, PHP ha sido uno de los leguajes de script más potentes y estables para generar páginas

web sobre servidores Linux e incluso Windows. PHP se integra de forma perfecta con bases de datos MySQL y servidores Apache.

1.15.2. JavaScript

Es el lenguaje utilizado del lado del cliente. Es interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C. Al igual que Java, JavaScript es un lenguaje orientado a objetos propiamente dicho, ya que dispone de herencia, si bien esta se realiza siguiendo el paradigma de programación basada en prototipos, ya que las nuevas clases se generan clonando las clases base (prototipos) y extendiendo su funcionalidad.

Javascript se puede incluir en cualquier documento HTML, o todo aquel que termine traduciéndose en HTML en el navegador del cliente; ya sea PHP, ASP, JSP, SVG.

Incluir código directamente en una estructura HTML es una práctica invasiva, y no recomendada. El método correcto que define la W3C es incluir javascript como un archivo externo, tanto por cuestiones de accesibilidad, como practicidad y velocidad en la navegación.

1.16. Control de versiones.

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (por ejemplo, para algún cliente específico).

El control de versiones se realiza principalmente en la industria informática para controlar las distintas versiones del código fuente. Sin embargo, los mismos conceptos son aplicables a otros ámbitos como documentos, imágenes, sitios web.

1.16.1. Subversion.

Es un sistema de control de versiones libre y de código fuente abierto, es decir, maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. En este aspecto, mucha gente piensa en los sistemas de versiones como en una especie de máquina del tiempo. Subversion puede acceder al repositorio a través

de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer que la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único si se ha hecho un cambio incorrecto a los datos, simplemente se deshace ese cambio (16). Subversion es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros. Para usted, esos ficheros pueden ser código fuente, cualquier cosa desde la lista de la compra de comestibles hasta combinaciones de video digital y más allá (16).

1.16.2. TortoiseSVN.

Es un cliente gratuito de código abierto para el sistema de control de versiones Subversion. Maneja ficheros y directorios a lo largo del tiempo. Los ficheros se almacenan en un repositorio central. El repositorio es prácticamente lo mismo que un servidor de ficheros ordinario, salvo que recuerda todos los cambios que se hayan hecho a sus ficheros y directorios. Esto permite que pueda recuperar versiones antiguas de sus ficheros y examinar la historia de cuándo y cómo cambiaron sus datos, y quién hizo el cambio. Es fácil de usar, permite ver el estado de los archivos desde en el explorador de Windows y permite también crear gráficos de todas las revisiones asignadas.

1.16.3. Rapid SVN y KDESVN.

Clientes gráficos para el Subversion que permiten su integración con el sistema en los distintos Sistemas Operativos Linux.

1.17. Conclusiones.

Con el desarrollo de este capítulo se puede concluir que para realizar la Arquitectura de Sistema se necesita aplicar tanto los estilos arquitectónicos, así como patrones de diseño para la solución de problemas que se presenten, además de que se necesita de una serie de actividades para la realización de la arquitectura; todo esto es imprescindible para garantizar una arquitectura estable, que cumpla con los estándares y normas definidas por el proyecto con lo cual se logra un sistema de mayor calidad y que cumpla con las necesidades de los clientes. De los estudios realizados el estilo arquitectónico que se va a utilizar es el Modelo Vista Controlador, el

cual fue orientado por la línea de arquitectura del proyecto, además de los diferentes patrones de diseño y arquitectónicos. También se utilizará la programación orientada a aspectos por las facilidades que brinda la misma. Además se decidió que para el mejor desarrollo de este trabajo es necesario realizar un conjunto de actividades que garanticen la calidad del resultado final. Las Herramientas a utilizar por definición del marco de trabajo, serán: Servidor web Apache 2.0 o superior, PHP 5.2 o superior, con los siguientes módulos o extensiones: pdo,pdo_pgsql, pgsql. soap, Gestor de Base de Datos PostgreSQL 8.3 o superior, Cliente o Explorador Web de la familia Mozilla, preferentemente Firefox u otro que implemente el DOM 2.0 y que soporte JavaScript, cliente SVN para descargar actualizaciones del repositorio,TorstoiseSVN para Windows, RapidSVN para Linux,IDEs de desarrollo para PHP y/o JavaScript: ZendStudio Neon para PHP y para ZendFramework, Eclipse, agregar pluing para PHP y pluing para JavaScript (Spket o Aptana).

CAPÍTULO II: PROPUESTA ARQUITECTÓNICA

2.1. Introducción.

En este capítulo se hace una propuesta de arquitectura de sistema de la línea Planificación Empresarial y Presupuestada, con el objetivo de darle respuesta al problema que se plantea en el diseño teórico. Además se realizarán una serie de artefactos para la realización de la misma.

2.2. Descripción General de la Arquitectura de Planificación Empresarial y Presupuestada.

La arquitectura del sistema de Planificación Empresarial y Presupuestada está compuesta por un conjunto de componentes en los cuales agrupan los requisitos funcionales para dar una solución de forma genérica a la planificación ya sea a nivel empresarial o a nivel de entidades presupuestadas.

2.3. Arquitectura del negocio.

La arquitectura del negocio se realiza en la etapa de modelado del negocio. Esta puede verse como un conjunto de elementos relacionados entre sí por medio de una funcionalidad determinada que representa la estructura y la organización de un “Sistema de Negocio”. Incluye, a un nivel de abstracción alto, la descripción de los procesos y estructuras básicas del negocio. Durante la realización de la misma en la línea Planificación Empresarial y Presupuestada se realizaron tres actividades fundamentales, la confección del modelo conceptual en conjunto con los analistas y los desarrolladores y el equipo de arquitectura, el agrupamiento de los requerimientos funcionales por componentes, y la priorización de estos requisitos y componentes para organizar su implementación en dependencia de la complejidad de cada uno durante todo el proceso de desarrollo de la aplicación.

2.3.1. Modelo conceptual

El modelo conceptual es una idea global sobre los individuos, los grupos, las situaciones y los acontecimientos que interesan a una disciplina. Los modelos conceptuales se construyen a partir de los conceptos que son palabras que describen imágenes mentales de los fenómenos, y de las proposiciones que establecen las relaciones entre ellos. Por tanto un modelo conceptual es un grupo de conceptos y de juicios que lo integran dentro de una configuración. Proporcionan una organización para pensar, observar e interpretar lo que se ve. Ofrecen igualmente una orientación para identificar las cuestiones más importantes de cada fenómeno y para dar soluciones a los problemas que se presentan en la práctica.

Además proporcionan un criterio general para poder descubrir cuando un problema ha sido solucionado.

Explica los conceptos significativos en el dominio del problema. La definición de modelo conceptual ofrece la ventaja de subrayar frecuentemente una concentración en los conceptos del dominio, no en las entidades del software. Puede mostrar:

- Conceptos
- Asociaciones
- Atributos de conceptos

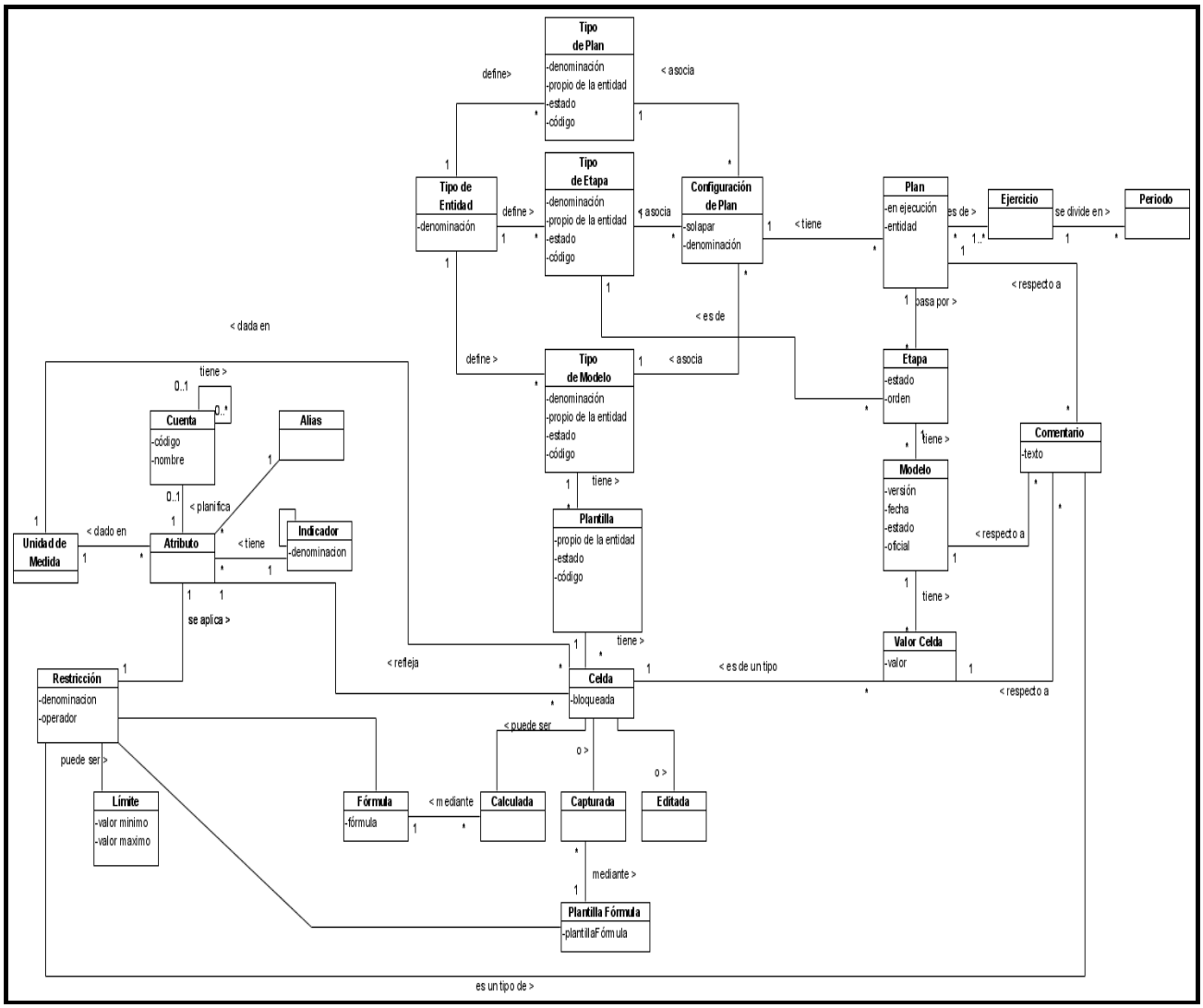


Figura 3: Modelo Conceptual

Dentro del modelo conceptual los principales conceptos identificados fueron:

- ✓ Ejercicio
- ✓ Plan
- ✓ Etapa
- ✓ Plantilla
- ✓ Modelo

- ✓ Celda
- ✓ Restricción

Para la mejor comprensión del modelo conceptual consultar el diccionario de datos, ver Anexo 2.

2.3.2. Agrupamiento de los requerimientos funcionales por componente

Cada uno de los componentes que se definió por parte del equipo de soluciones agrupa un conjunto de funcionalidades que fueron identificadas por los de analistas de la línea Planificación Empresarial y Presupuestada quedando agrupadas como muestra la siguiente tabla:

Tabla 3: Requerimiento funcionales

Componente	Agrupación de requisitos	Requisitos
Nomencladores	Gestionar nomencladores	➢ Adicionar elemento de nomenclador ➢ Modificar elemento de nomenclador ➢ Eliminar elemento de nomenclador ➢ Activar elemento de nomenclador ➢ Desactivar elemento de nomenclador ➢ Consultar elemento de nomenclador
Indicadores	Gestionar indicadores	➢ Adicionar indicador ➢ Modificar indicador ➢ Eliminar indicador ➢ Consultar indicador
Plantilla	Gestionar plantilla	➢ Adicionar plantilla ➢ Modificar plantilla ➢ Eliminar plantilla ➢ Activar plantilla ➢ Desactivar plantilla ➢ Consultar plantilla
	Gestionar columnas	➢ Adicionar columna

		➢ Eliminar columna
	Gestionar filas	➢ Adicionar fila ➢ Eliminar fila
	Configurar celdas	➢ Configurar celda calculada
	Gestionar nombre de columna	➢ Adicionar nombre de columna ➢ Activar nombre de columna ➢ Desactivar nombre de columna ➢ Consultar nombre de columna
Configuración de plan	Gestionar asociaciones	➢ Adicionar configuración ➢ Eliminar configuración ➢ Activar configuración ➢ Desactivar configuración ➢ Consultar configuración
Construcción de plan	Gestionar plan	➢ Adicionar plan ➢ Definir etapas asociadas
	Configurar celdas de modelo	➢ Configurar celdas de modelo
	Gestionar modelos	➢ Adicionar modelo
Realización de plan	Gestionar restricciones	➢ Adicionar restricción ➢ Eliminar restricción ➢ Consultar restricción ➢ Validar restricción
	Gestionar comentario	➢ Adicionar

		comentario ➢ Modificar comentario ➢ Consultar comentario
	Oficializar plan	➢ Oficializar plan
	Gestionar modelos	➢ Modificar modelo ➢ Eliminar modelo ➢ Consultar modelo ➢ Terminar modelo ➢ Confirmar modelo ➢ Rechazar modelo ➢ Hacer oficial ➢ Copiar modelo ➢ Calcular celdas
	Gestionar plan	➢ Eliminar plan ➢ Consultar plan
Procesador matemático	Gestionar fórmulas	➢ Adicionar fórmula ➢ Modificar fórmula ➢ Eliminar fórmula ➢ Consultar fórmula
	Validar fórmula	➢ Validar fórmula
	Obtener valores de argumentos de la fórmula	➢ Obtener valores de argumentos de la fórmula
	Obtener expresión polaca	➢ Obtener expresión polaca
	Evaluar función	➢ Evaluar función

2.3.3. Priorización de los requisitos y componentes.

Para llevar a cabo la priorización de componentes, se agruparon requisitos de cada uno y se les determinó cuantitativamente una complejidad y una prioridad para el negocio, y esto permitió determinar el factor de esfuerzo (suma de la complejidad del requisito, complejidad integración del mismo, la cantidad de entidades asociadas, la cantidad de gestoras y la cantidad de tablas) y la prioridad final del requisito (suma del factor de esfuerzo y de la prioridad del negocio). Luego, la complejidad del componente no es más que la suma de las

prioridades finales de los requisitos que este agrupa. Los componentes se priorizan de mayor a menor, o sea, los más complejos deben implementarse primero (17). Con el objetivo de lograr una mayor comprensión de lo antes expuesto aparecen a continuación las fórmulas utilizadas para el desarrollo de esta actividad, ver Anexo 5.

Prioridad final del componente = ∑Factor de esfuerzo + ∑Prioridad Negocio

Factor de esfuerzo = CR + CI + CE + CG+CT

CR = complejidad del requisito

CI = complejidad integración del requisito

CE = cantidad de entidades asociadas

CG = cantidad de gestoras

CT = cantidad de tablas

Tabla 4: Priorización de los componentes

	Componentes	**Complejidad**
1-	Realización del plan	427
2-	Cálculo de necesidades	286
3-	Plantilla	227
4-	Nomencladores	157
5-	Configuración del plan	128
6-	Construcción del Plan	108
7-	Procesador matemático	103
8-	Indicadores	97
9-	Buscador	33

2.4. Integración Horizontal.

El sistema del proyecto ERP Cuba está dividido en 11 módulos, o subsistemas principales con los cuales se integra el Subsistema de Planificación Empresarial y Presupuestada. Por la complejidad en la integración debido a que el subsistema interactúa con la mayoría de los subsistemas del proyecto la línea de Arquitectura implementó un componente utilizando el

patrón Service Locator[5] para que la integración se realice a través del mismo, es decir, los indicadores necesarios se obtendrán a través de dicho componente (ver figura 3). Los subsistemas de los cuales el componente de Arquitectura toma los diferentes indicadores son:

- Contabilidad
- Costos y procesos
- Inventario
- Facturación
- Activos Fijos Tangibles e Intangibles
- Caja y Banco
- Cobros y Pagos
- Planificación Empresarial y Presupuestada
- Capital Humano
- Estructura y Composición
- Configuración

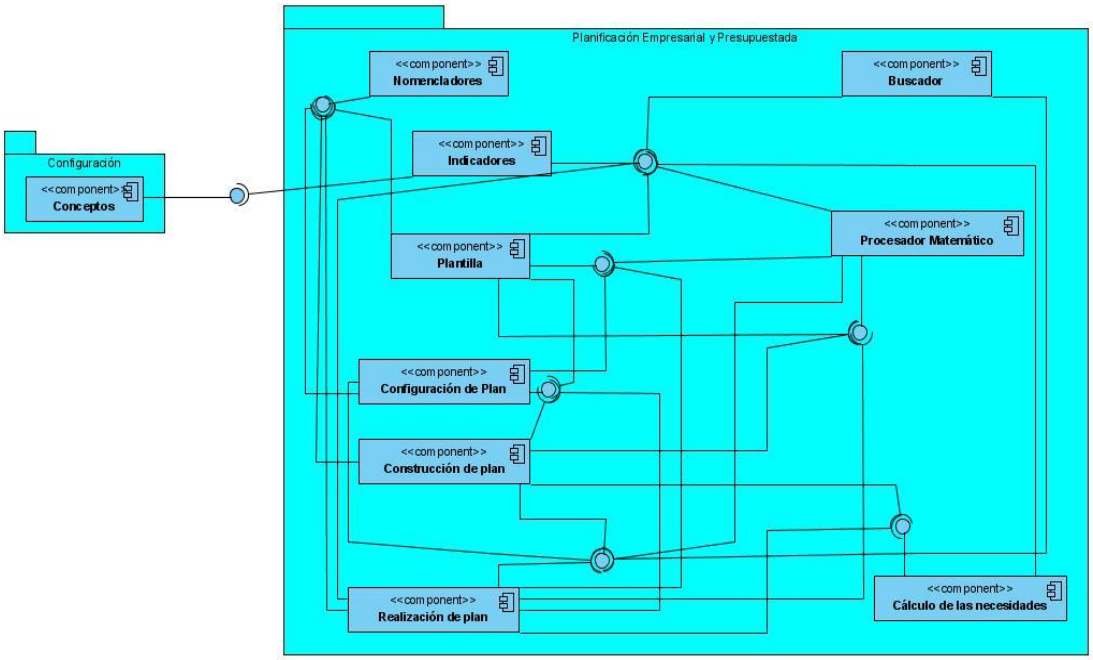


Figura 4: Integración horizontal de la línea Planificación Empresarial y Presupuestada

El Subsistema de Planificación Empresarial y Presupuestada esta compuesto por 9 componentes principales, estos componentes son:

- Nomencladores
- Configurar plan

[5] El **Service Locator** abstrae las dependencias del vendedor, las complejidades de la búsqueda, y la creación de objetos de negocio, y proporciona un interface simple para los clientes. Esto reduce la complejidad del cliente. Además, el mismo cliente y otros clientes pueden reutilizar el **Service Locator**. Este patrón proporciona un mecanismo para abstraer todas las dependencias y detalles de red dentro del Service Locator.

- Indicadores
- Plantilla
- Construcción del Plan
- Realización del plan
- Cálculo de necesidades
- Procesador matemático
- Buscador

Estos se integran horizontalmente en cada una de las capas. En la capa de Presentación se integran mediante el Portal UCID. En la de negocio la integración es mediante el IoC[6]. En la capa de Datos la integración se lleva a cabo a través de sus respectivas interfaces.

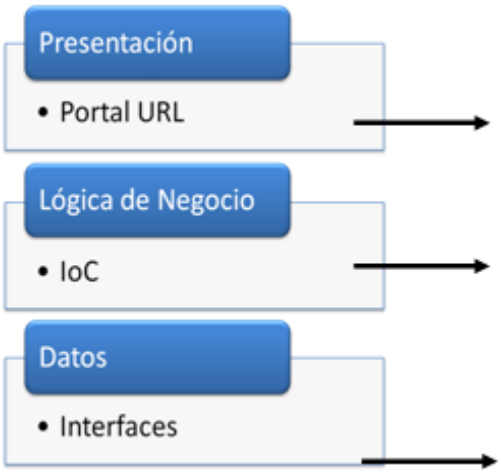


Figura 5: Integración horizontal de la arquitectura

2.5. Integración Vertical.

La arquitectura usa la AOP[7], o sea, Programación Orientada a Aspectos, pues ayuda a modificar dinámicamente el modelo estático al incluir el código requerido para cumplir los requerimientos secundarios sin tener que modificar el modelo estático original.

Algunos de los aspectos tratados son:

[6] **Inversión de control** (*Inversion of Control* en inglés, **IoC**) es un concepto junto a unas técnicas de programación en las que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales, en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos (procedure calls) o funciones.

[7] La **Programación Orientada a Aspectos** (AOP) es un paradigma de programación relativamente reciente cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos. Gracias a la AOP se pueden encapsular los diferentes conceptos que componen una aplicación en entidades bien definidas, eliminando las dependencias entre cada uno de los módulos. De esta forma se consigue razonar mejor sobre los conceptos, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables.

- Validación
- Excepciones
- Transacciones
- Trazas
- Concurrencia
- Caché
- Conceptos Globales

La integración vertical se realiza mediante Doctrine en la capa de datos, Zend y Zend EXTjs en la lógica del negocio, y EXTjs en la capa de presentación.

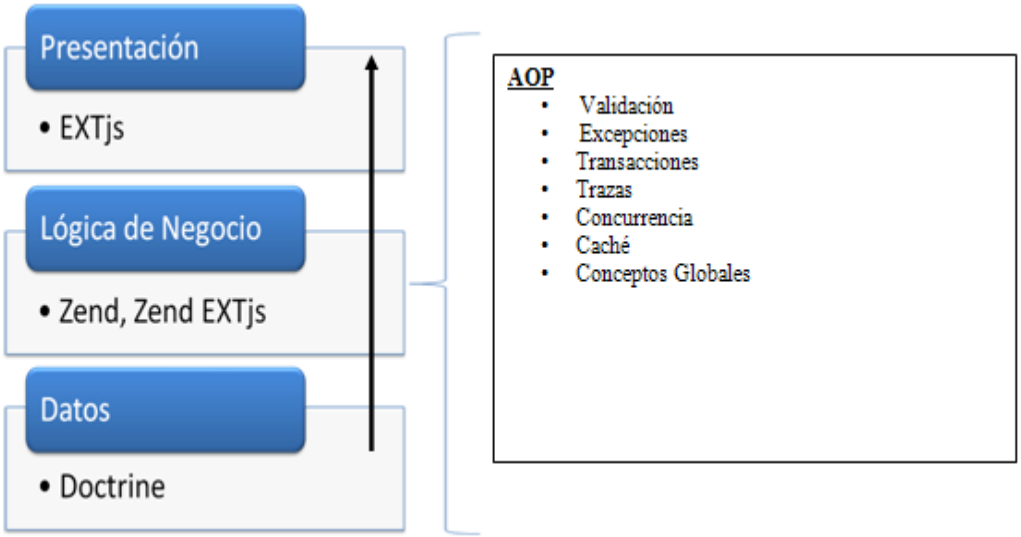


Figura 6: Integración vertical de los componentes.

2.5.1. **Concurrencia.**

Este componente realizado por la línea de arquitectura del proyecto, se pone de manifiesto en el ejemplo de modificar información, en las tablas de la base de datos, hay un campo llamado versión que no se muestra en la interfaz pero que se almacena en el formulario, y que se obtiene y pasa en la consulta cuando se ejecuta la modificación. Este campo se crea y se trata con el objetivo de realizar un control de concurrencia, pues se puede dar el caso de que dos personas extraen la información para modificar al mismo tiempo y entonces dichos datos llegan a la interfaz de usuario con una misma versión, pero si una persona modifica primero los datos se incrementará la versión de la fila y entonces cuando la segunda persona desee modificar la información y envíe la petición como tendrá una versión menor a la que existirá en el momento en la base de datos se le muestra un mensaje indicándole que actualice

nuevamente la información y luego la modifique, ya que en el momento en que la estaba observando dicha información fue modificada y por lo tanto aún no ha visto dichos datos (18).

2.5.2. **Transacciones.**

Componente con el que cuenta el framework y mediante el cual serán salvados automáticamente los datos de modificaciones e inserciones. El mismo realizará la apertura, cierre y marcha atrás de las transacciones. Favorece el hecho de que las funcionalidades de modificación no necesiten crear instancias de la clase modelo ni ejecuten métodos en la misma, sino que solamente extraigan los datos de los campos de la presentación y con esto ya el framework hace la salva de dichos datos en la base de datos. Le facilita al programador la implementación, puesto que anula la necesidad de crear consultas de inserción, producto a esto solo se debe programar la obtención de los campos de la presentación y el framework será el responsable de que los mismos sean guardados en la base de datos. Este componente es un aspecto del sistema y en su configuración se realiza un registro como aspecto para especificar si se quiere activar o no la salva automática (18).

2.5.3. **Validación en el servidor.**

Es un aspecto del componente de programación orientada a aspectos (AOP) y se encarga de validar los campos que serán introducidos en inserciones a la base de datos, de manera que siempre sea comprobado que estén correctos los mismos antes de introducirlos en las tablas. A través del mismo se disparan las excepciones del subsistema (18).

2.5.4. **Excepciones.**

Con la implementación y puesta en marcha del manejador de excepciones no será necesario realizar el lanzamiento de excepciones en cada funcionalidad de las clases, solamente se deberá registrar el método y la excepción en el xml que lleva por título managerexception.xml, a través de este y del denominado exception se realizará todo el tratamiento necesario para cuando se dispare una excepción en el framework (18).

2.5.5. **AOP (Programación Orientada a Aspectos).**

Este componente se encuentra en la carpeta de recursos comunes del proyecto, en los xml se encuentran 3 que componen dicho elemento y sus nombres son aspect, aspecttemplate y weaver.

Aspect: Registra todos los aspectos necesarios a cargar por la aplicación y lleva asociado tres aspectos esenciales:

- *Validation:* Validación del lado del servidor como aspecto del sistema.

- *Salvamodelos:* Especifica el trabajo con las transacciones, o sea la ejecución de las salvas automáticas de datos a la base de datos.
- *Interpreterrules:* Este aspecto se utiliza para interpretar las reglas del negocio que se guardan en la base de datos para este subsistema, este componente evalúa reglas establecidas con anterioridad.

Aspecttemplate: Plantilla del aspecto que registran los aspectos y se indica si son aspectos precondicionales, postcondicionales o en caso de fallas registra el aspecto validation e indica que es precondicional.

Weaver ó Tejedor**:** Este fichero finalmente especifica cuando se ejecuta un determinado método la plantilla de aspectos que se va a cargar, la misma cargará los aspectos en específicos que tiene registrados (18).

2.5.6. **Traza.**

La traza no es más que un registro de determinadas acciones tanto de los usuarios como del sistema, el componente se va a componer de dos elementos básicos, un lanzador de eventos que muestran una acción especificada al usuario y un manejador de trazas que va a encargarse de registrar dichos historiales ya sea en bases de datos o en algún fichero en específico (18).

2.5.7. **Caché.**

Este elemento del framework será el encargado de poner en la memoria cache del explorador los elementos que sean necesarios (18).

2.6. Modelo arquitectónico.

El paquete de Servicios incluirá todas las clases y funcionalidades contenidas en los paquetes de Controladoras, Modelo, y Validaciones. Para solicitar un servicio que está en otro dominio, se accede a través del paquete de Servicios, quien analizará la solicitud e irá a la clase que tiene dicha funcionalidad y la devolverá a Servicios que a su vez se la entregará al dominio solicitante. Además este paquete de Servicios también tiene lo denominado “Lógica adicional” que consiste en que a la hora de que se haga una solicitud que no esté implementada como tal, pero que necesita de los parámetros que tienen los paquetes mencionados, se implementa esa solicitud en este paquete (Servicios) y devuelve el resultado al servicio (19).

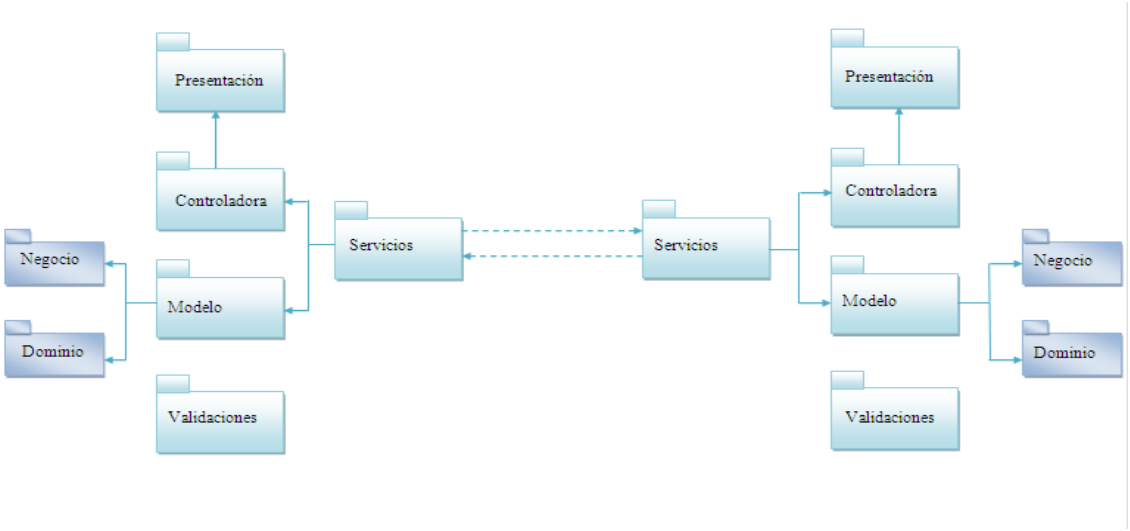


Figura 7: Estructura de los componentes

2.7. Presentación de los componentes.

El subsistema está dividido en 9 componentes agrupando estos un conjunto de funcionalidades y requerimientos específicos del cliente. Los mismos interactúan y comparten datos en dependencia de la funcionalidad de cada uno. Esta actividad por la complejidad que presenta para su realización se obtuvo del análisis de los requisitos y casos de uso definidos en el negocio, también de los modelos conceptuales que se conformaron. Los componentes definidos se presentan a continuación:

Figura 8: Gráfico Estructural de la Arquitectura del Sistema Planificación

2.8. Responsabilidad arquitectónica de los componentes.

Nomencladores: Este componente agrupa tres nomencladores que van a ser utilizados en la línea (tipo de plan, tipo de etapa, tipo de modelo).

Indicadores: Los indicadores pueden ser tanto externos como internos. Este componente es el encargado de gestionar los indicadores propios de la línea así como de agrupar los que se encuentran en los demás subsistemas del proyecto, el mismo interactúa como se observó anteriormente con el componente de la línea de arquitectura del proyecto para realizar la búsqueda de los indicadores que procedan de las demás líneas del ERP.

Configurar plan: En este componente se crean las configuraciones que van a tener los planes posteriormente, en las mismas se realizan las diferentes asociaciones que son necesarias para crear el plan, a un tipo de plan se le asocian las etapas por las que pasará, por cada etapa definida para el plan se debe especificar los tipos de modelos que se llenarán en cada una de dichas etapas.

Construcción del Plan: Este componente es el encargado de crear el plan con los datos propios del mismo, se definirá para cuales ejercicios va a estar vigente el plan que se está creando, se cargarán todas las configuraciones existentes ,es decir, las que se crearon en el componente Configurar Plan , se escogerá una de estas, de la escogida, se listan todas las etapas asociadas a esta configuración y se pasa a brindar la opción que el usuario establezca

un orden entre estas etapas y crean los pertinentes modelos por cada uno de los tipos de modelos que presenta ya el plan en cada una de sus etapas, además se clasifican las celdas. Se debe permitir que el usuario, luego de haber especificado la plantilla que va a tener asociada cada uno de los modelos, pueda clasificar las celdas editadas en operacional o capturada.

Realización del plan: En este componente se van a gestionar las restricciones a nivel de plan, además se llenarán los modelos que tiene el mismo, se asociarán las restricciones a cada celda del modelo que lo lleve, se entran los datos en las celdas editadas y luego se dará aplicar, se mandan a calcular cada una de las fórmulas que existan tanto en celdas calculadas, operacional como capturadas, también se validarán las restricciones que pesan sobre aquellas celdas determinadas, se cambiará el estado de los modelos, los cuales tienen cuatro estados, elaboración, confirmado, rechazado y terminado, por último se realizará el cierre de las etapas.

Cálculo de necesidades: En este componente se gestionan las normas de consumo y de productos, así como los niveles de actividades asociados a cada una con el objetivo de hacer el cálculo de necesidades para conocer el total de recursos que se deben solicitar para llevar a cabo las actividades que se proponen en un plan.

Procesador matemático: Este componente es un asistente matemático para gestionar las fórmulas asociadas a las celdas de un modelo, así como las restricciones que pesen sobre dichas celdas.

Buscador: Este componente se encargará de buscar datos de varios elementos (planes, modelos, plantillas, indicadores).

2.9. Entradas y salidas de los componentes

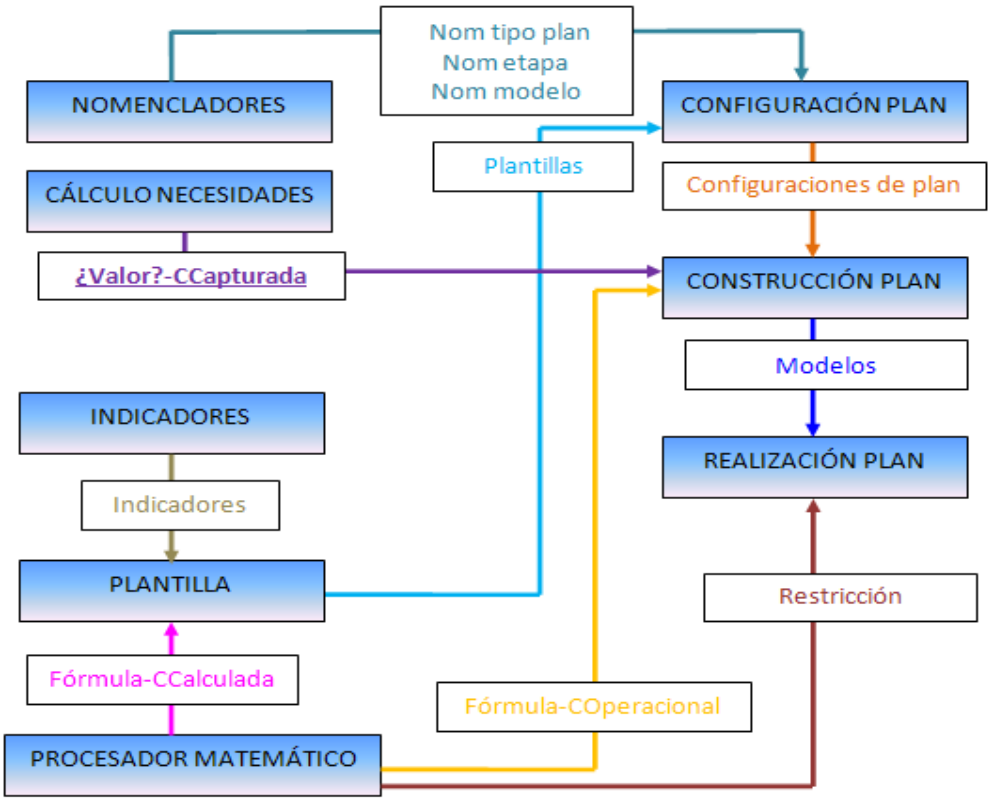


Figura 9: Entradas y salidas de los componentes del subsistema

La figura anterior muestra como es que se relacionan de forma general los componentes en el subsistema, es decir los objetos, fórmulas o valores mas importantes que salen y entran de cada unos de ellos.

Para un mejor entendimiento de esto la siguiente tabla muestra de forma detallada las entradas y las salidas de cada uno de los componentes del subsistema.

Tabla 5: Entradas y salidas de los componentes

Componente	Entra	Sale
Nomencladores		➢ Nomenclador tipo plan ➢ Nomenclador tipo etapa ➢ Nomenclador tipo modelo
Indicadores	➢ Indicadores externos	➢ Indicadores
Plantilla	➢ Indicadores	➢ Plantilla
Configuración de plan	➢ Nomenclador tipo plan	➢ Configuración de plan

	➢ Nomenclador etapa ➢ Nomenclador modelo	
Construcción de plan	➢ Configuración de plan ➢ Cálculo de necesidad (valor) ➢ Fórmula (celda operacional o capturada)	➢ Modelo
Realización de plan	➢ Modelo	➢ Plan
Procesador matemático	➢ Nomenclador tipo plan ➢ Nomenclador etapa ➢ Nomenclador modelo ➢ Plantillas	➢ Fórmula (celda operacional) ➢ Fórmula (celda calculada) ➢ Fórmula (celda capturada) ➢ Resultado de fórmulas
Cálculo de Necesidades		Cálculo de necesidad (valor)

2.10. Diagrama de componentes.

El diagrama de componentes muestra como se relacionan los mismos dentro del subsistema comunicándose a través de las diferentes interfaces que provee cada uno de ellos para así lograr la integración interna de la línea.

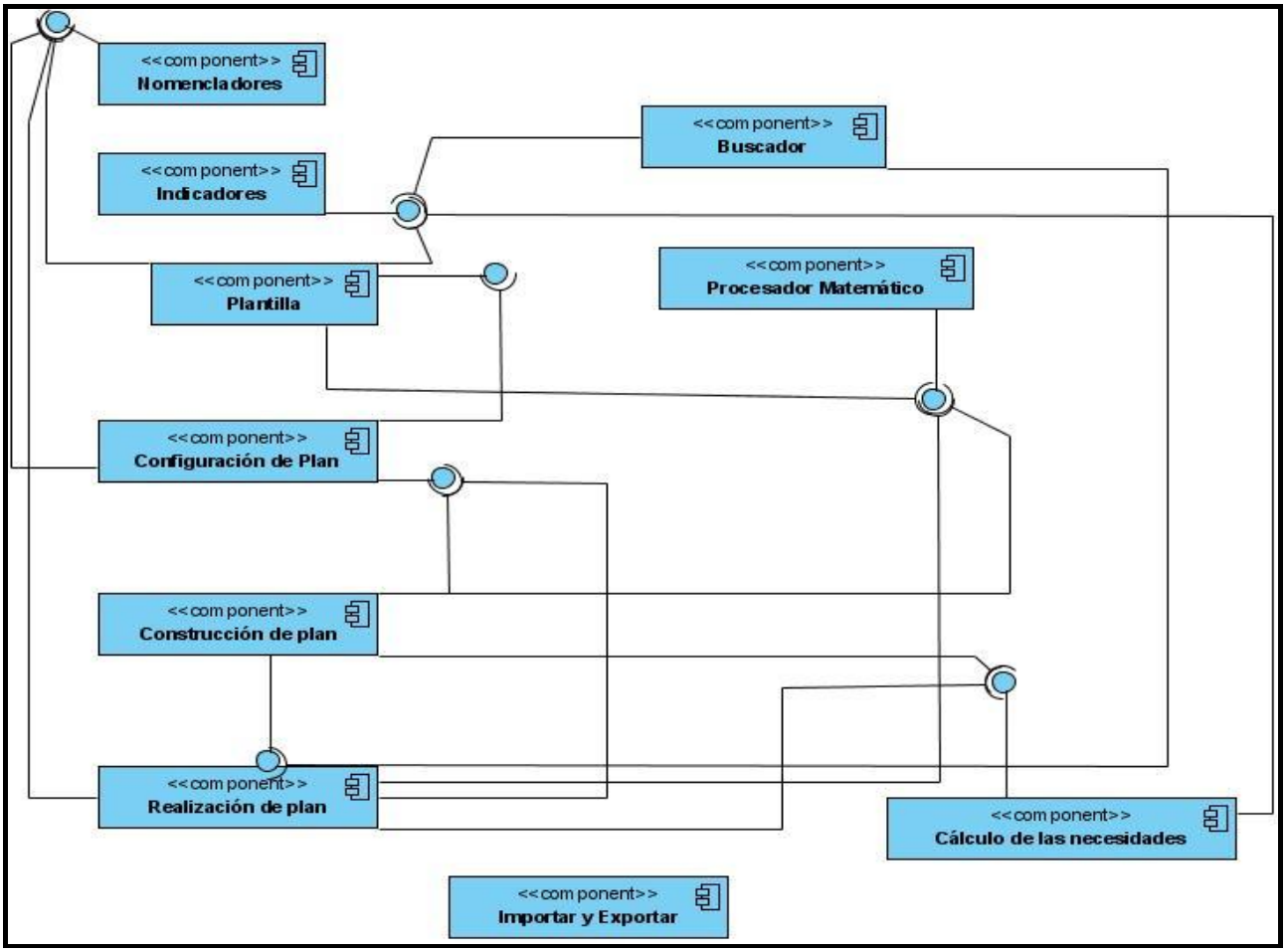


Figura 10: Diagrama de componentes

Además para controlar la integración interna de la línea se elaboró un documento en el cual se reflejan cada uno de los servicios que son brindados por las interfaces provistas de cada uno de los componentes, recogiendo la información de servicios por componente, una breve descripción del los mismos, así como quien lo usa y el valor que devuelve, para una mayor información remitirse al Anexo 6.

2.11. Elementos globales utilizados.

Las variables globales son elemento de alto uso para el sistema, en el caso de la línea de Planificación Empresarial y Presupuestada no se definen variables globales dentro de su implementación, sin embargo utiliza algunas de las que están definidas para todo el ERP, las mismas se muestran a continuación:

Tabla 6: Elementos globales

Necesidad	Proveedor	Descripción de la necesidad	Formato en que lo solicita

ejercicio	Configuración	Para obtener todos los ejercicios que existen en el sistema.	En forma de arreglo.
estructura	Estructura y Composición	Conocer el id de la estructura de entidad solicitante.	Formato del tipo definido (integer).

2.12. Diagramas de clases de diseño

Estos diagramas describen gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Para la elaboración de los mismos se tuvieron en cuenta los patrones Grasp y el patrón modelo vista controlador, siguiendo estos patrones se presentan los diagrama de clases.

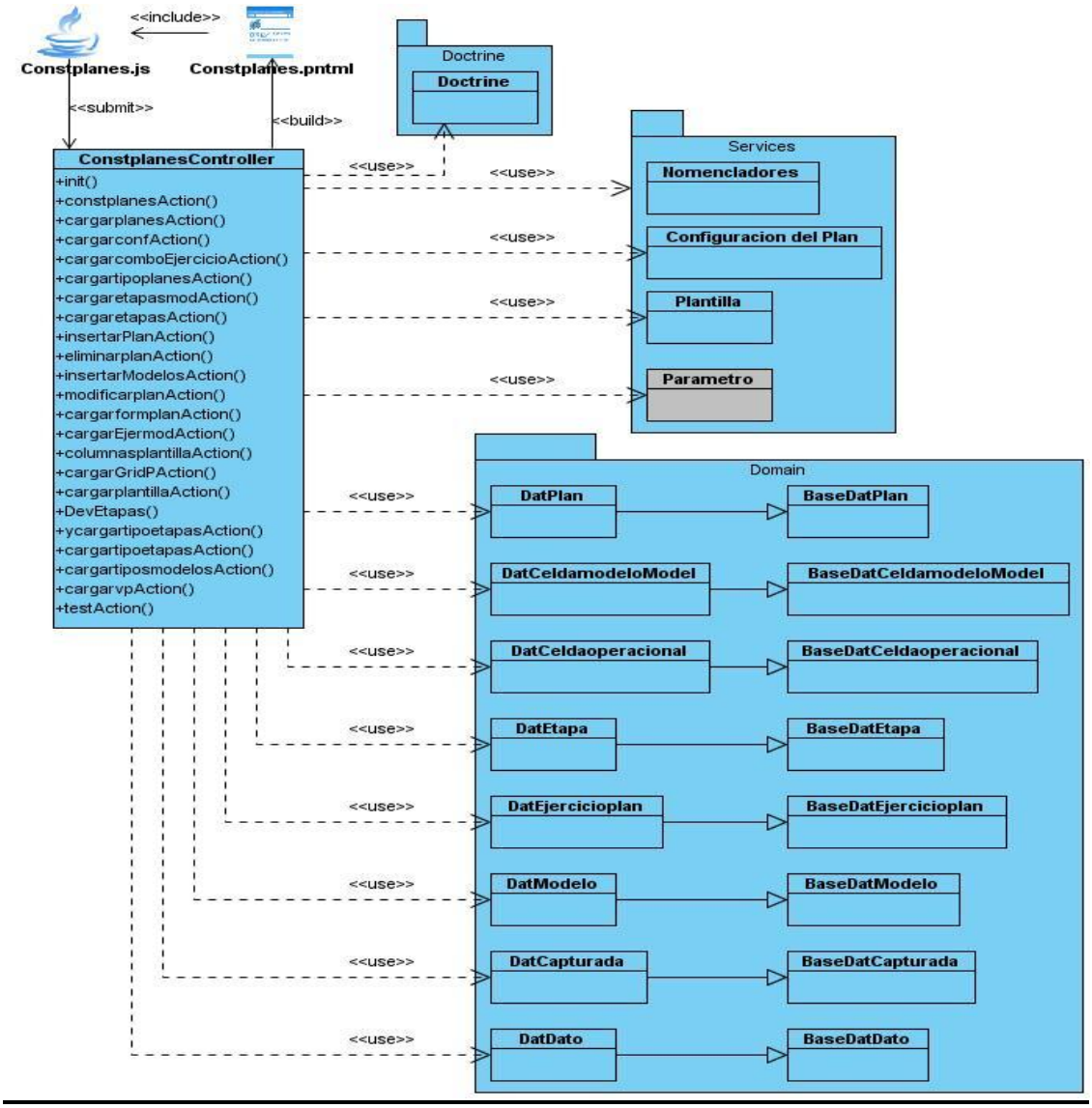


Figura 11: Diagrama de clases del componente Construcción del plan.

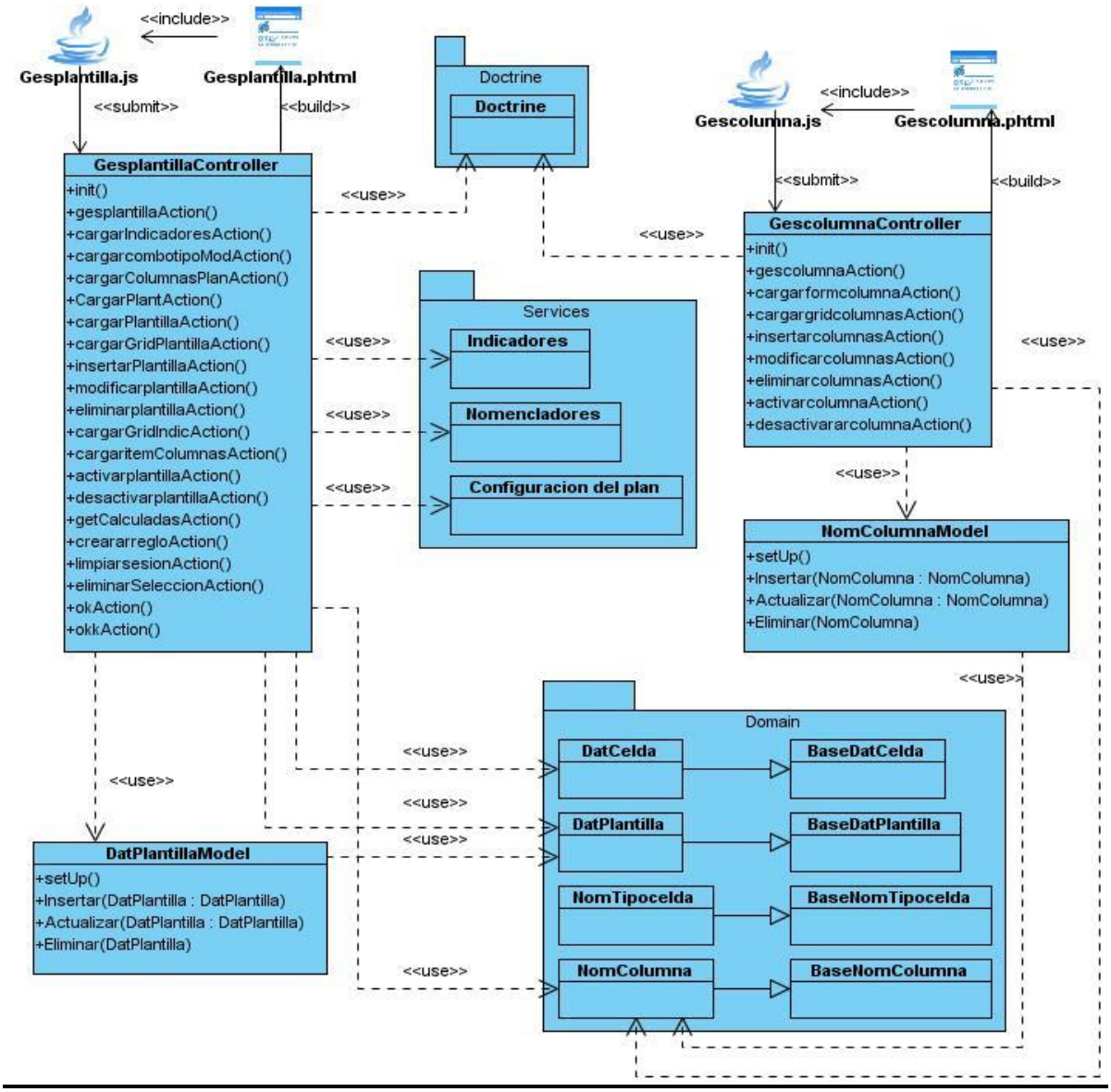


Figura 12: Diagrama de clases del componente Plantilla.

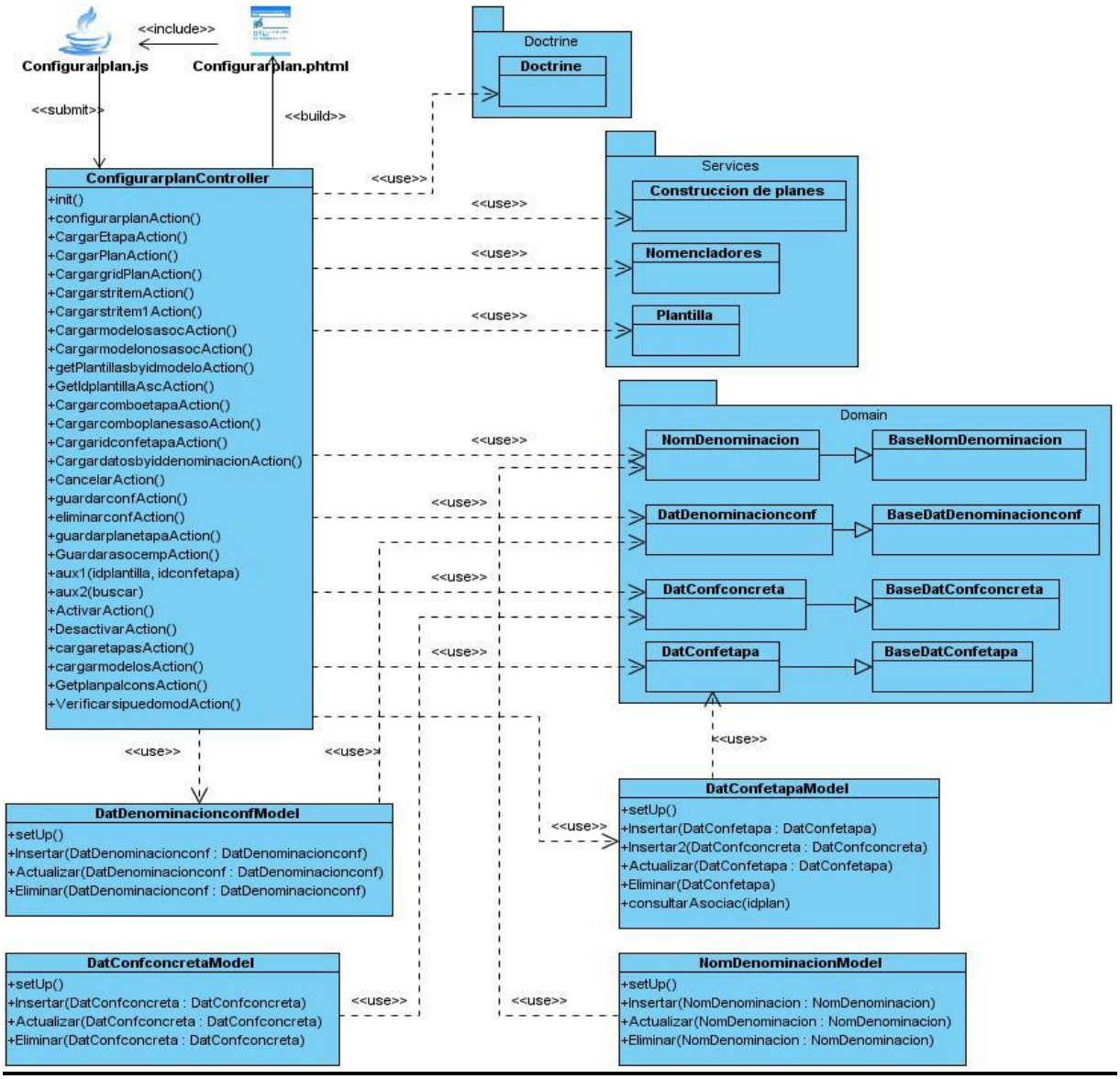


Figura 13: Diagrama de clases del componente Configuración del plan.

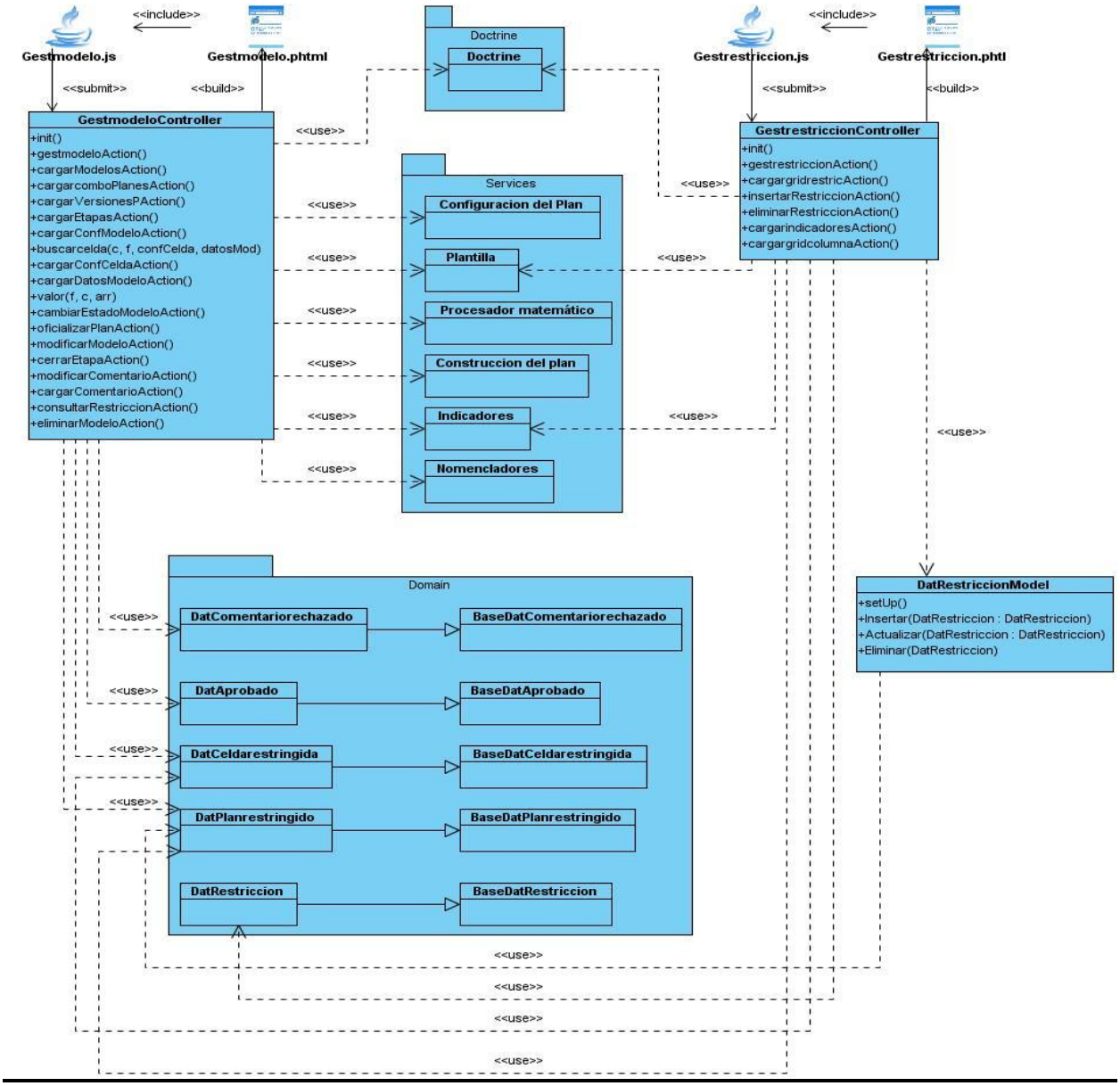


Figura 14: Diagrama de clases del componente Realización del plan.

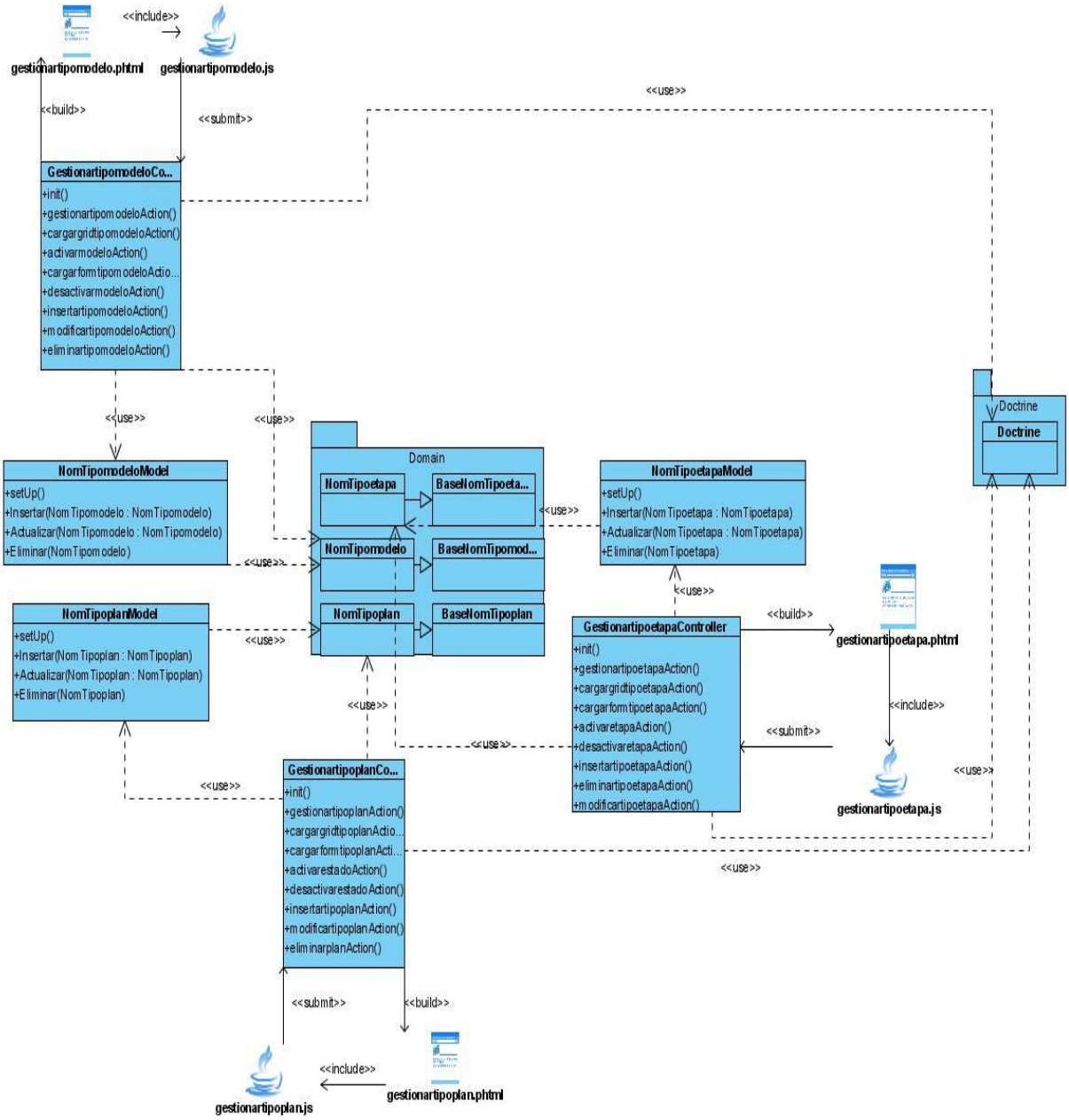


Figura 15: Diagrama de clases del componente Nomencladores.

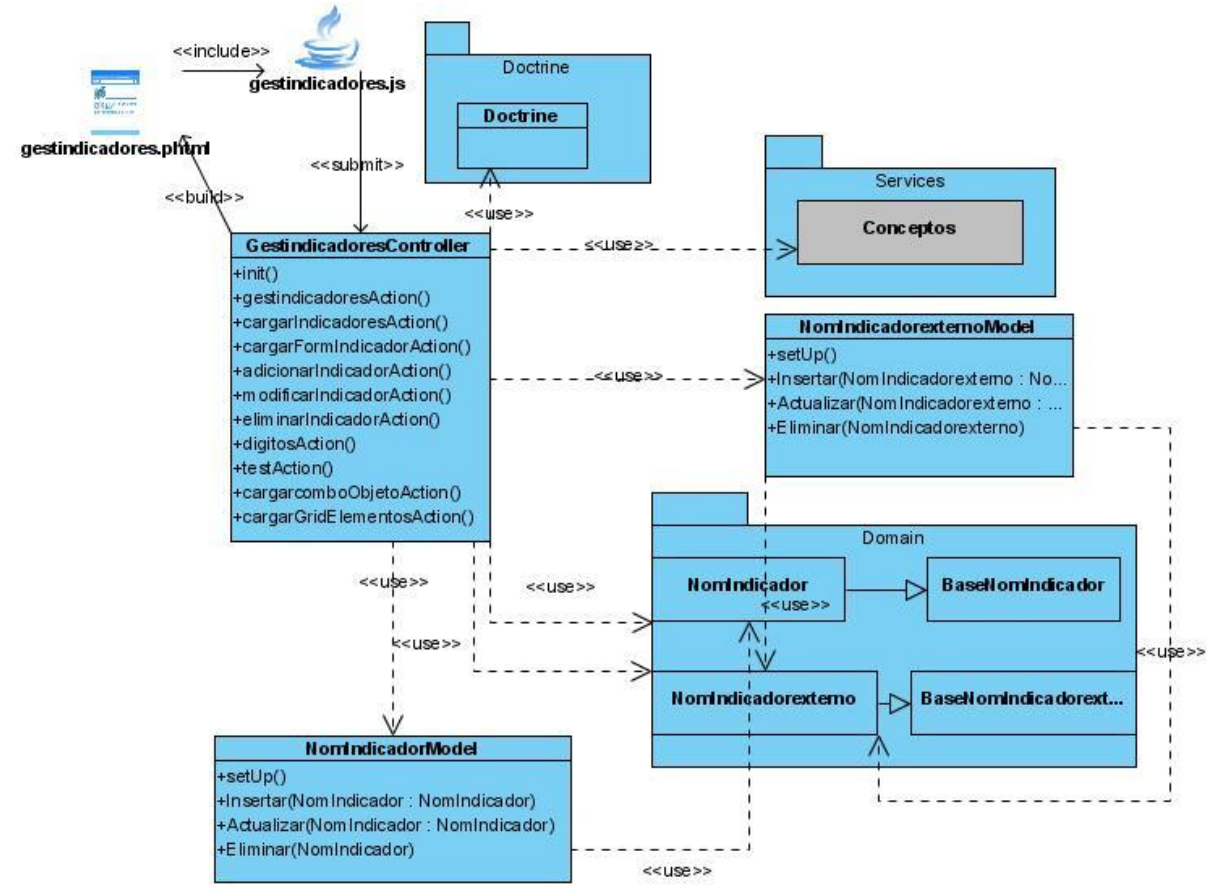


Figura 16: Diagrama de clases del componente Indicadores.

Para el caso del componente cálculo de las necesidades debido a la cantidad de gestores que presenta se hacía muy complejo realizar un solo diagrama de clases por lo que se decidió realizar uno por cada gestor que presentaba, los mismos aparecen a continuación:

Gestionar Actividad

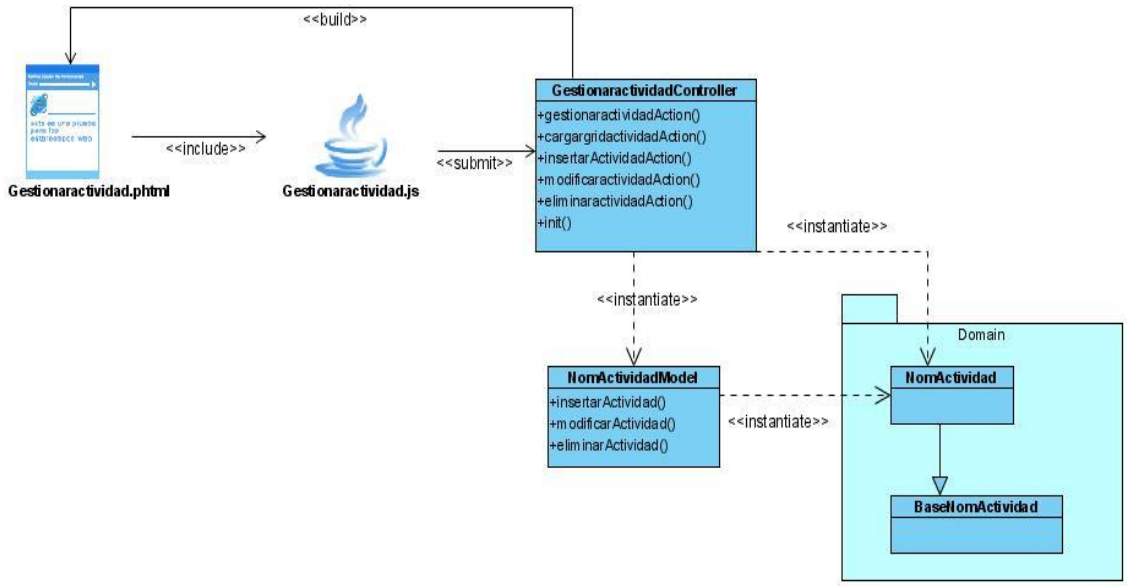


Figura 17: Gestionar Actividad

Gestionar Criterio

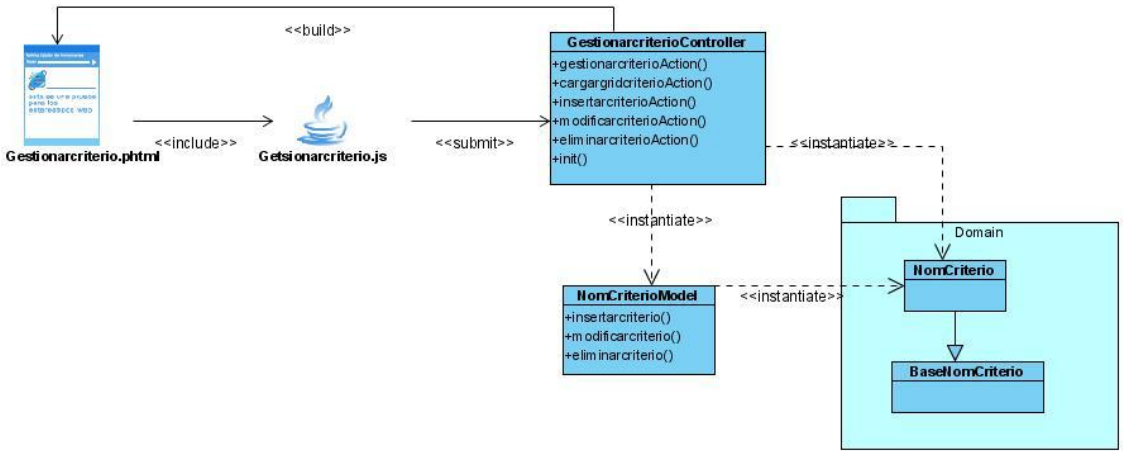


Figura 18: Gestionar Criterio

Gestionar Concepto

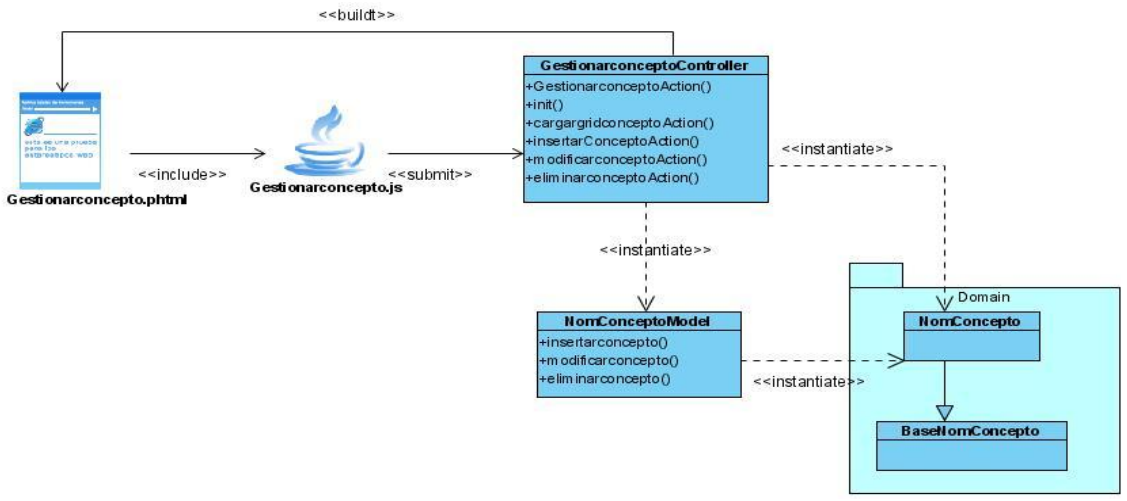


Figura 19: Gestionar Concepto

Gestionar asociación Actividad-Criterio

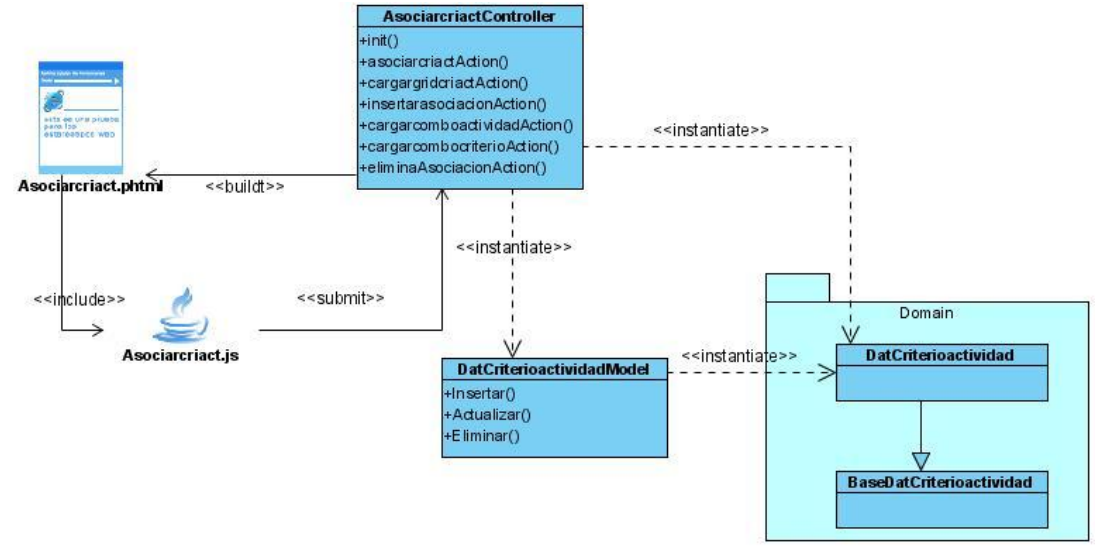


Figura 20: Gestionar asociación Actividad-Criterio

Gestionar asociación Criterio-Concepto

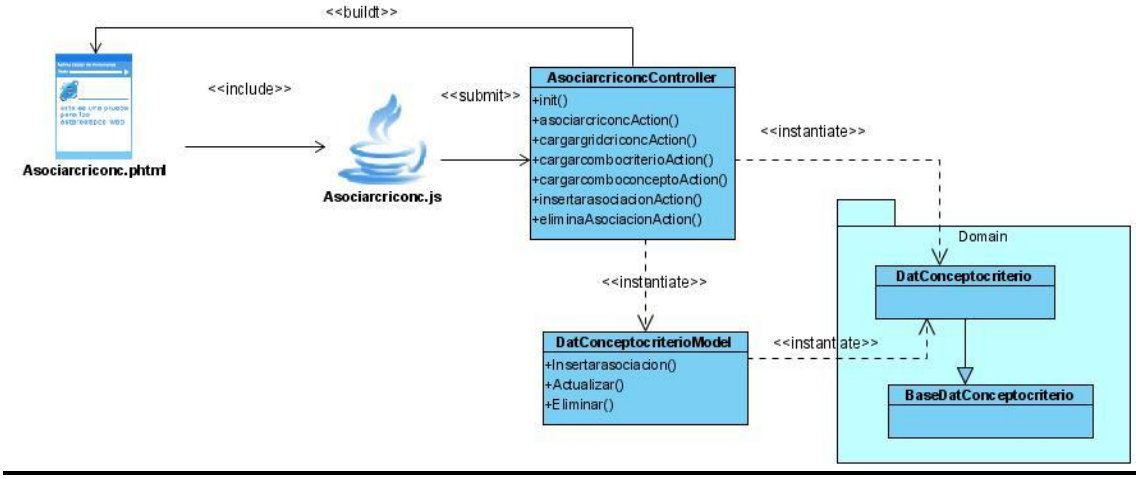


Figura 21: Gestionar asociación Criterio -Concepto

Gestionar niveles de actividad

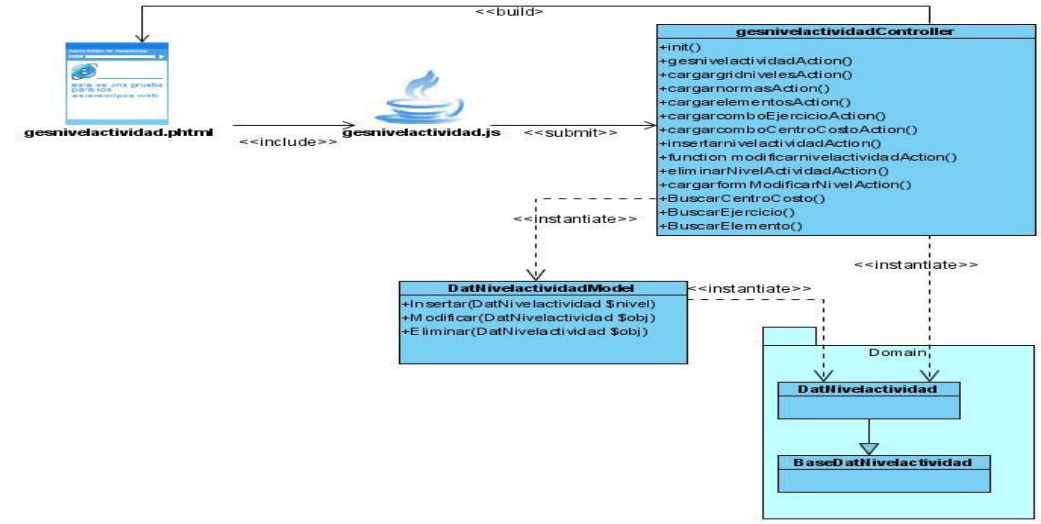


Figura 22: Gestionar nivel de actividad

Gestionar norma

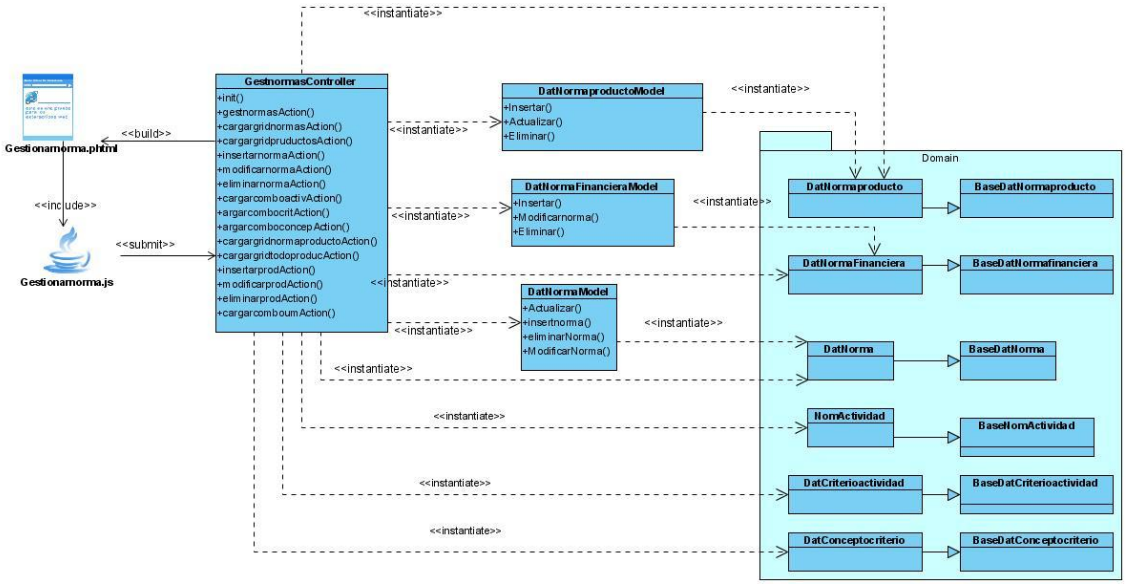


Figura 23: Gestionar norma

Calcular necesidades

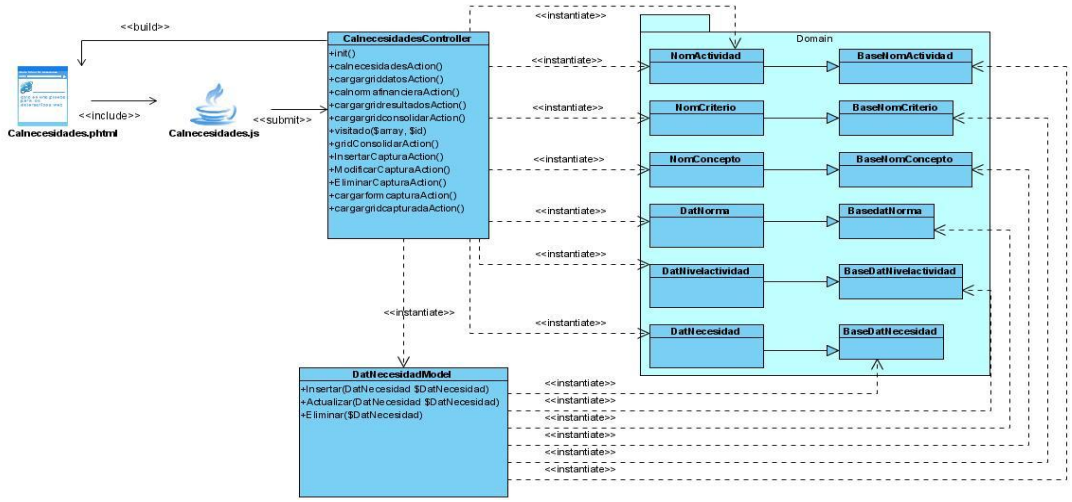


Figura 24: Calcular necesidades

2.13. Patrones de diseño implementados.

En los componentes definidos fueron aplicados los patrones GRASP Experto, Bajo Acoplamiento y Alta Cohesión. Se buscó que cada clase incluida fuese responsable de los aspectos que le conciernen, y tratando de tener la menor dependencia funcional y de datos posible con otras clases. Esto se pone de manifiesto en la creación de un gestor de negocio por cada proceso del mismo, como son los casos del flujo de Gestionar restricciones y el Gestor de Gestionar Plantilla, así como en el componente Cálculo de las Necesidades para la asociación de Actividades a Criterios y de Criterios a Conceptos interviene un Gestor especializado en el tema, además se logró un bajo acoplamiento ya que se obtuvo una

escasa dependencia entre las clases y aumentó la reutilización, esto se pone de manifiesto cuando se hace uso del IoC para obtener cualquier dato o colección de datos que se necesite desde un componente determinado.

También se implementó en el diseño de la solución el patrón ¨Fachada¨ que entra dentro de los patrones de diseño de categoría estructural, con el propósito de proporcionar una interfaz unificada de alto nivel, representando a cada uno de los componentes, y que facilite su uso, estas fachadas satisfacen a la mayoría de los componentes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas, como ejemplo de las mismas se tienen a:

- ConfplanServices.php
- ConstplanServices.php
- IndicadoresServices.php

Estas se utilizaron debido a que existen muchas dependencias entre los componentes y las clases que implementan una abstracción. Las mismas proporcionan a los componentes independencia y portabilidad.

Además en cada uno de los componentes del subsistema se utilizó también el patrón “Singleton “o solitario, este es utilizado para garantizar que una clase sólo tenga una única instancia, proporcionando un punto de acceso global a la misma. Esto se pone de manifiesto cuando en cada uno de los controladores al inicio dentro del constructor se llama la función render (), con la cual se accede a la vista, la clase que implementa esta funcionalidad esta implementada en el framework, sin embargo todos los subsistemas del ERP la utilizan, debido a que el acceso de una instancia única es más controlado, se reduce el uso de variables globales.

Conjuntamente con los patrones anteriores se utilizó el Chain of Responsability o cadena de responsabilidades, el mismo se utilizó para proporcionar a más de un objeto la capacidad de atender una petición, para así evitar el acoplamiento con el objeto que hace la petición. Se forma con estos objetos una cadena, en la cual cada objeto o satisface la petición o la pasa al siguiente.

2.14. Integración de sistema y datos.

En el subsistema los componentes se encuentran relacionados cada uno con diferentes tablas del modelo de datos (ver Anexo1) con las cuales van a interactuar y realizar sobre las mismas las diferentes transacciones, en la tabla se muestran solo las principales a realizar por cada uno de los componentes.

Tabla 7: Integración de sistema y datos

Componente	**Tabla con la que interactúa**	**Principales transacciones realizadas**
Nomencladores	nom_tipomodelo	Adicionar, modificar, eliminar y activar o desactivar el modelo.
	nom_tipoplan	Adicionar, modificar, eliminar y activar o desactivar el tipo de plan.
	nom_tipoetapa	Adicionar, modificar, eliminar y activar o desactivar el tipo de etapa.
Plantilla	nom_tipocelda	
	nom_columna	Adicionar, modificar, eliminar y activar o desactivar la columna.
	dat_celda	Adicionar celda
	dat_plantilla	Adicionar, modificar, eliminar y activar o desactivar la plantilla.
indicadores	nom_indicadorexterno	
	nom_indicador	Adicionar, modificar, eliminar un indicador.
procesador_matematico		
construccion_planes	nom_estadomodelo	
	dat_modelo	Adicionar modelo
	dat_etapa	Adicionar etapa
	dat_plan	Adicionar plan
	dat_ejercicioplan	Adicionar plan
	nom_version	
	nom_versionplan	Adicionar versión del plan
	nom_versionplanmodelo	Adicionar versión del planmodelo
	dat_celdamodelo	Adicionar celdamodelo

	dat_celdaoperacional	Adicionar celda operacional
	dat_capturada	Adicionar celda capturada
realizacion_de_plan	dat_celdarestringida	Adicionar celda restringida
	dat_restriccion	Adicionar y Eliminar una restricción.
	dat_planrestringido	Adicionar plan restringido
	dat_aprobado	Adicionar plan aprobado
	dat_comentariorechazado	Adicionar comentario
configurar_plan	nom_denominacion	Adicionar denominación
	dat_denominacionconf	Adicionar y Eliminar
	dat_confconcreta	Adicionar configuración concreta
	dat_confetapa	Adicionar configuración de etapa
calculo_de_necesidades	dat_necesidad	Adicionar necesidad
	dat_nivelactividad	Adicionar, Modificar y Eliminar un nivel de actividad determinado.
	dat_normaproducto	Adicionar, Modificar y Eliminar una norma producto.
	dat_norma	Adicionar, Modificar y Eliminar una norma.
	dat_normafinanciera	Adicionar, Modificar y Eliminar una norma.
	nom_actividad	Adicionar, modificar, eliminar una actividad.
	nom_criterio	Adicionar, modificar, eliminar un criterio.
	nom_concepto	Adicionar, modificar, eliminar un concepto.
	dat_criterioactividad	Adicionar y Eliminar asociación.
	dat_criterioconcepto	Adicionar y Eliminar asociación.

2.15. Línea base.

La arquitectura que se presenta define la línea que deben seguir los procesos de análisis, diseño e implementación, evaluando las mejores prácticas y estilos vigentes en la actualidad referentes a la Arquitectura de software que permita a los desarrolladores y demás involucrados tener una idea clara de lo que se está implementando. La línea base permitirá guiar y evaluar el desarrollo del proyecto, según las normas definidas por la Arquitectura, y definir los elementos de configuración.(14)

Componentes:

1. Componente Nomencladores
   1.1. Gestionar nomencladores
      R1. Adicionar elemento de nomenclador
      R2. Modificar elemento de nomenclador
      R3. Eliminar elemento de nomenclador
      R4. Activar elemento de nomenclador
      R5. Desactivar elemento de nomenclador
      R6. Consultar elemento de nomenclador

2. Componente Indicadores
   2.1. Gestionar indicadores
      R7. Adicionar indicador
      R8. Modificar indicador
      R9. Eliminar indicador
      R10 Activar indicador
      R11. Desactivar indicador
      R12. Consultar indicador

3. Componente Plantilla
   3.1. Gestionar Plantilla
      R13. Adicionar plantilla
      R14. Modificar plantilla
      R15. Eliminar plantilla
      R16. Activar plantilla
      R17. Desactivar plantilla

R18. Consultar plantilla

3.2. Gestionar columna

R19. Adicionar columna

R20. Eliminar columna

3.3. Gestionar filas

R21. Adicionar fila

R22. Eliminar fila

3.4 Configurar celdas

R23. Configurar celda calculada

4. Componente Configuración de plan

4.1. Gestionar asociaciones

R24. Adicionar configuración

R25. Eliminar configuración

R26. Activar configuración

R27. Desactivar configuración

R28. Consultar configuración

5. Componente Construcción de plan

5.1. Gestionar plan

R29. Adicionar plan

R30. Definir etapas asociadas

5.2. Configurar celdas del modelo

R31. Configurar celdas del modelo

5.3. Gestionar modelos

R32. Adicionar modelo

6. Componente Realización de plan

6.1. Gestionar restricciones

R33. Adicionar restricción

R34. Eliminar restricción

R35. Consultar restricción

R36. Validar restricción

6.2. Gestionar comentario

R37. Adicionar comentario

R38. Modificar comentario

R39. Consultar comentario

6.3. Gestionar modelos

R40. Modificar modelo
R41. Eliminar modelo
R42. Consultar modelo
R43. Consultar modelo
R44. Terminar modelo
R45. Confirmar modelo
R46. Rechazar modelo
R47. Hacer oficial
R48. Duplicar modelo
R49. Calcular celda

6.4. Oficializar plan
R50. Oficializar plan

6.5. Gestionar plan
R51. Eliminar modelo
R52. Consultar modelo
R53. Iniciar Planificación

7. Componente Procesador matemático
7.1. Gestionar fórmula
R54. Adicionar fórmula
R55. Modificar fórmula
R56. Eliminar fórmula
R57. Consultar fórmula

7.2. Validar fórmula
R58. Validar fórmula

7.3. Obtener valores de argumentos de fórmula
R59 Obtener valores de argumentos de fórmula

7.4. Obtener expresión polaca
R60. Obtener expresión polaca

7.5. Evaluar función
R61. Evaluar función

8. Componente Cálculo de necesidades
8.1. Gestionar actividad
R62. Adicionar actividad
R63. Modificar actividad
R64. Eliminar actividad

8.2. Gestionar criterio

R65. Adicionar criterio

R66. Modificar criterio

R67. Eliminar criterio

8.2. Gestionar concepto

R68. Adicionar concepto

R69. Modificar concepto

R70. Eliminar concepto

8.3. Gestionar asociación conceptos a criterios

R71. Adicionar asociación

R72. Eliminar asociación

8.4. Gestionar asociación criterios a actividades

R73. Adicionar asociación

R74. Eliminar asociación

8.5. Gestionar niveles de actividad

R75. Adicionar nivel de actividad

R76. Modificar nivel de actividad

R77. Eliminar nivel de actividad

8.6. Gestionar normas

R78. Adicionar norma

R79. Modificar norma

R80. Eliminar norma

R81. Mostrar norma

8.7. Gestionar valores capturados

R82. Capturar valores

R83. Modificar valores capturados

R84. Eliminar valores capturados

8.7. Convertir UM

R85. Convertir UM

8.8. Consolidar necesidades

R86. Consolidar necesidades

8.9. Calcular necesidades

R87. Calcular necesidades

2.16. Matriz de Integración.

Matriz de Relaciones									
Entradas/Salidas	**Nomencladores**	**Indicadores**	**Plantilla**	**Configuración d**	**Construcció**	**Realización del**	**Cálculo**	**Procesador matem**	**Buscador**
Nomencladores									
Indicadores			DatCelda			DatCeldarestringida			
Plantilla	NomTipomodelo	NomIndicador		DatConfconcreta				Zend_Session_Namespace	
Configuración del plan	NomTipoplan NomTipoetapa NomTipomodelo		DatPlantilla		DatPlan				
Construcción del plan	NomTipoplan NomTipoetapa NomTipomodelo	NomIndicador	DatCelda DatPlantilla	DatDenominacionconf NomDenominacion DatConfetapa				Zend_Session_Namespace	
Realización del plan	NomTipomodelo NomTipoetapa	NomIndicador	NomColumna DatCelda DatPlantilla	DatConfetapa	DatModelo DatVersionplan DatEtapa DatPlan DatDato			PolacaModel	
lculo de las necesidad									
rocesador matemátic		NomIndicador	DatCelda		DatModelo DatEjercicioplan				
Buscador		NomIndicador	DatPlantilla		DatModelo DatVersionplan DatPlan				

Figura 25: Matriz de relaciones entre componentes.

La figura anterior muestra cuales son las clases que devuelven los datos de los servicios que necesita el componente en cuestión, por ejemplo el componente Plantilla necesita de un servicio que le devuelva todos los indicadores para la confección de la misma, la clase que devolverá dichos indicadores será NomIndicador y así sucesivamente con los demás componentes como se observan.

2.17. Convenciones o estándares de códigos.

Las convenciones y estándares de código proporcionan un lenguaje común en el equipo de desarrollo y entre aplicaciones (20). Con el objetivo de lograrlo y atendiendo al campo de acción de este trabajo se propone que se usen las establecidas en los lineamientos de arquitectura propuestos por la línea de Arquitectura del proyecto (Normas y estándares de Codificación del ERP).

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. Los estándares de codificación en el marco del ERP van a permitir una mejor integración entre las líneas de producción y se establecerán las

pautas que conlleven a lograr un código más legible y reutilizable, de tal forma que se pueda aumenta su mantenibilidad a lo largo del tiempo (21).

2.18. Estándares de diseño.

La consecución de los objetivos perseguidos a través de la puesta a disposición del público de cualquier aplicación Web está condicionada por la satisfacción del usuario final, con el propósito de lograr un estándar de diseño que satisfaga las necesidades y exigencias de los clientes, sea compatible con las tecnologías utilizadas y que cumpla con todos los objetivos deseados. El trabajo en equipo es uno de los requisitos fundamentales para que una interfaz responda a las características que se exigen, es importante que toda el área implicada siga una misma línea, que se cumplan con las reglas y las responsabilidades del trabajo en cuestión. Los estándares utilizados por la línea de Planificación Empresarial y Presupuestada serán los que siguen una política estándar a cumplir por todas las aplicaciones desarrolladas en el centro UCID, los mismos se encuentran en el siguiente documento (Estándar de diseño de interfaces para las aplicaciones de gestión) (22).

2.19. Plan de Iteración.

El Plan de Iteración recoge la planificación detallada de un período corto de tiempo dentro del proyecto – una iteración. Entre otras cosas, debe identificar las actividades, los riesgos involucrados, los artefactos a actualizar o crear, las necesidades de adquisición de bienes o servicios y los hitos esperados durante la iteración (23).

El objetivo de este plan es definir detalladamente para cada una de las iteraciones a realizarse un conjunto de tareas, actividades y recursos por cada uno de los componentes existentes en la línea, por tal motivo existirá para cada iteración del ciclo de vida del proyecto un artefacto de este tipo. Para cada iteración existe una serie de objetivos los cuales son usados como referencia de evaluación para determinar diferentes aspectos, como grado de terminación de una determinada función, rendimiento, niveles de calidad.

Para cada iteración de los componentes fue necesario detallar la programación estimada para la iteración, así como porciento de terminación de cada una de las tareas a realizar en la implementación, los recursos a emplear, la cantidad de funcionalidades que va a tener el componente y escenarios que van ser tomados en cuenta y finamente se deben establecer los criterios de evaluación que se van a tener para la iteración. Es recomendable para las

iteraciones emplear herramientas para la planeación de proyectos con el fin de hacer mas fácil y organizada esta tarea (21), en este caso se utilizó la herramienta MS Project.

Este plan está compuesto por toda la información necesaria para llevar a cabo la implementación de los componentes. Es utilizado por la dirección del proyecto para dirigir las actividades a realizar durante el proceso de desarrollo del software (24). En los anexos se muestra una figura con el plan de iteración, así como un ejemplo de cómo en la línea se llevo a cabo la implementación de uno de los componentes, las actividades a realizar por cada uno de los requerimientos que agrupaba el mismo para su implementación. Anexo 3 y Anexo 4.

2.20. Conclusiones

En este capítulo se realizó la propuesta de la arquitectura de sistema de la línea Planificación Empresarial y Presupuestada. Se analizó la arquitectura del negocio realizando para ello un modelo conceptual, agrupando los requerimientos funcionales por componentes y luego priorizando los mismos para luego dar paso a la realización de los diferentes artefactos que eran necesarios en este trabajo.

CAPÍTULO III: EVALUACIÓN DE LA ARQUITECTURA

3.1. Introducción.

Al final del desarrollo del software se conoce si éste cumplió o no con al menos un atributo de calidad que se especificaron en los requerimientos no funcionales, lo que implica tomar demasiados riesgos innecesarios. Para evitar estos riesgos es recomendable realizar evaluaciones a la arquitectura durante su diseño.

Realizar un buen diseño de arquitectura de software garantiza que el sistema cumpla con uno o varios atributos de calidad, este diseño puede determinar el éxito o el fracaso de un sistema de software.

3.2. Evaluación de la arquitectura de software.

Uno de los factores que determina el éxito o el fracaso de software es su arquitectura, es decir, la estructura del sistema. Si la arquitectura ha sido bien diseñada, entonces, garantiza que el sistema cumpla con varios atributos de calidad, como por ejemplo confiabilidad, seguridad.

3.2.1. ¿Por qué es necesario evaluar una arquitectura de software?

Evaluar una arquitectura de software sirve para prevenir todos los posibles desastres de un diseño que no cumple con los requerimientos de calidad y para saber que tan adecuada es la arquitectura de software diseñada para el sistema. Una evaluación de una arquitectura no da un sí o un no, si es buena o mala, o una calificación, expresa donde está el riesgo, es decir, fortalezas y debilidades identificadas de la arquitectura.

Después de la evaluación de una arquitectura de software, se pueden tomar algunas decisiones cómo: si se puede seguir el proyecto con las áreas de debilidad dadas en la evaluación, si hay que reforzar la arquitectura de software o si hay que comenzar de nuevo toda la arquitectura.

3.2.2. ¿Cuándo es recomendable evaluar la arquitectura?

La evaluación clásica de la arquitectura se realiza cuando ésta se encuentra especificada totalmente y no se ha iniciado su implementación. La arquitectura puede ser evaluada en cualquier momento de desarrollo.

Existen dos variaciones útiles para realizar esta evaluación, la evaluación temprana y la evaluación tardía (25).

- **La evaluación temprana**

No es necesario que la arquitectura esté completamente especificada para efectuar la evaluación, y esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo (25).

- **La evaluación tardía**

  Cuando ésta se encuentra establecida y la implementación se ha completado. Este es el caso general que se presenta al momento de la adquisición de un sistema ya desarrollado.

Reglas de oro para determinar el momento de realizar evaluaciones (26):

- ✓ Realizar una evaluación cuando el equipo de desarrollo inicia a tomar decisiones que afectan directamente a la arquitectura.
- ✓ Cuando el costo de no tomar estas decisiones podría tomar más que el costo de realizar una evaluación.

3.2.3. ¿Quiénes participan en la evaluación?

Generalmente, las evaluaciones son realizadas por miembros del equipo de desarrollo, por ejemplo el arquitecto, el diseñador y el administrador de proyecto. Aunque pueden existir excepciones, donde las pruebas son realizadas por un grupo de especialistas.

El cliente también se interesa por las evaluaciones, pues en dependencia de los resultados obtenidos durante las pruebas puede decidir si se continua o no con el proyecto (26).

3.2.4 Resultado de la evaluación.

La evaluación de la arquitectura produce un informe, el cual varía en dependencia del método utilizado. Este informe produce respuesta a dos tipos de preguntas.

- ¿Es esta arquitectura adecuada para el sistema para el cual fue diseñada?
- ¿Cual de las arquitecturas propuestas es la más adecuada para el sistema?

Una arquitectura es adecuada cuando cumple dos criterios:

- El sistema resultante cumple con los atributos de calidad.
- El sistema puede ser construido con los recursos disponibles, es decir, es construible.

La evaluación de la arquitectura no produce resultados cuantitativos, no es de interés conocer la cantidad de transacciones por segundos debido a que el sistema aún no está implementado. Lo que interesa es aprender como un atributo de calidad es afectado por una decisión de diseño arquitectónico.

3.2.5 ¿Por qué cualidades puede ser evaluada una arquitectura?

Los atributos de calidad son requerimientos adicionales del sistema que hacen referencia a características que éste debe satisfacer.

Algunos de los atributos por los cuales puede ser evaluada una arquitectura son:

- Disponibilidad (Availability). Es la medida de disponibilidad del sistema para el uso.
- Desempeño (Performance). Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria. (IEEE 610.12).
- Confiabilidad (Reliability). Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo.
- Seguridad (Security). Es la mediada de la habilidad del sistema para resistir a intentos de uso no autorizados y negación de servicios, mientras se sirve a usuarios legítimos.
- Portabilidad (Portability). Es la habilidad del sistema de ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.

3.3. Técnicas de evaluación.

Existen varias técnicas que permiten realizar evaluaciones a la arquitectura, estas se clasifican en cualitativas y cuantitativas.

En las técnicas de evaluación cualitativas se pueden utilizar escenarios, cuestionarios o listas de verificación. Mientras que en las técnicas de evaluación cuantitativas se pueden emplear métricas, simulaciones, prototipos, experimentos o modelos matemáticos.

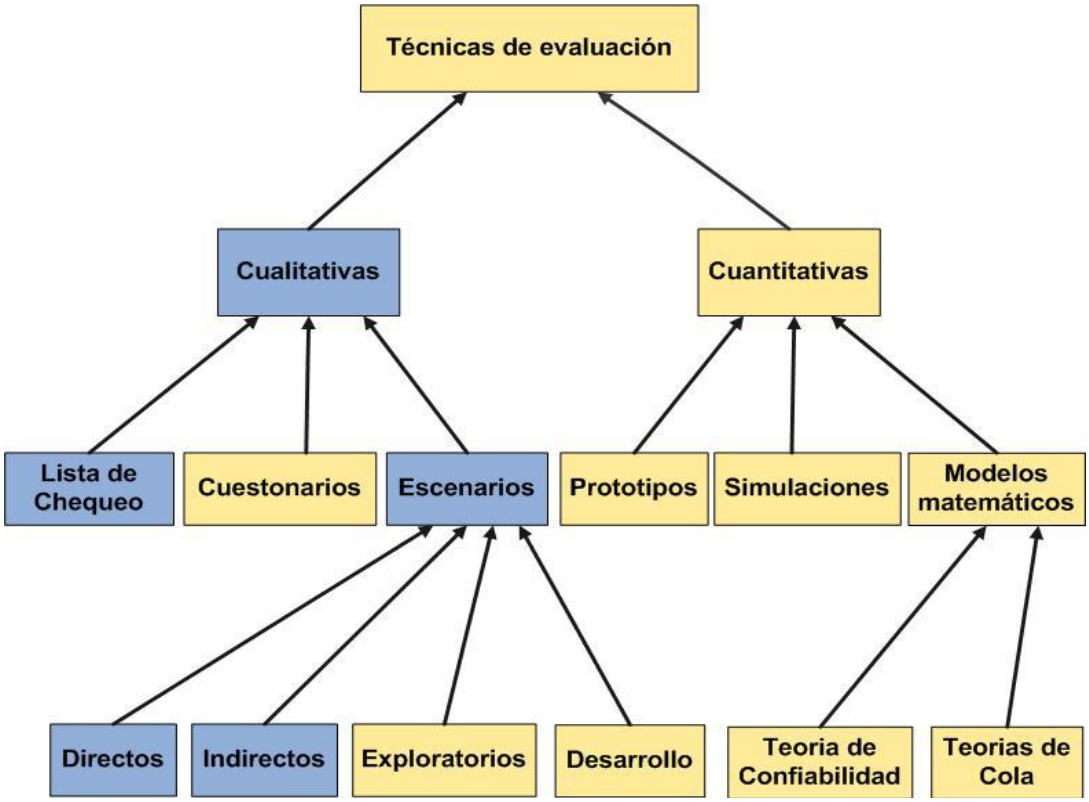


Figura 26: Técnicas de evaluación.

La mayoría de los métodos de evaluación utilizan escenarios, que son secuencias específicas de pasos que involucran el uso o la modificación del sistema. Por lo regular, las técnicas de evaluación cualitativas son usadas cuando la arquitectura se encuentra en construcción, mientras que las técnicas de evaluación cuantitativas, se usan cuando la arquitectura ya ha sido implantada.

3.4. Evaluación de la arquitectura de software propuesta.

La solución de arquitectura que se propone en este trabajo se encuentra especificada a alto nivel, es decir, un diseño poco detallado. Esta especificación a alto nivel impide la selección de escenarios que permitan validar si la arquitectura diseñada satisface a requisitos específicos. Como resultado de esto se realizará la evaluación atendiendo a como responden algunos elementos ante los principales atributos de calidad. Para ello se utilizó el método de evaluación ATAM (Analysis Tradeoff Architecture Method).

3.5. Método de Análisis de Desventajas de Arquitectura (ATAM). Principales características.

ATAM permite determinar el impacto de los atributos de calidad en el arquitectura, además provee un entendimiento interno, de cómo interactúan los principales objetivos de calidad.

CAPÍTULO III: EVALUACIÓN DE LA ARQUITECTURA

Cuando se evalúa usando ATAM, el objetivo es entender las consecuencias de las decisiones arquitecturales que respectan a los requerimientos de los atributos de calidad del sistema.
ATAM permite determinar si los objetivos concebidos se pueden alcanzar por la arquitectura propuesta. El método de análisis de arquitectura usado por ATAM permite que el análisis se pueda repetir. Además permite asegurar que las preguntas fundamentales respecto a los temas de arquitectura puedan ser respondidas tempranamente durante los requerimientos y los escenarios de diseño, cuando ocurre algún problema puede ser solucionado con bajos costos. Además provee a los involucrados encontrar un conflicto y la solución en la arquitectura de software.

¿Cuál es el propósito de ATAM?

Evaluar las consecuencias de las decisiones arquitecturales tomadas en consideración a los atributos de calidad requeridos. ATAM es un método para identificar riegos, esto significa que se detecta áreas de riesgos potenciales dentro de la arquitectura de un complejo sistema de software. Esto tiene algunas implicaciones:

- ATAM debe ser ejecutado tempranamente en el ciclo de desarrollo del software.
- Debe ejecutarse relativamente con un bajo costo y rápido, ya que evalúa los artefactos del diseño arquitectónico.
- ATAM producirá un análisis en proporción con el nivel de detalle de la especificación de la arquitectura. Además no siempre cuando se obtiene un análisis detallado de los atributos de calidad medibles de un sistema. (ej. latencia) puede ser el éxito. En cambio el éxito es lograr identificar las amenazas potenciales para el sistema.

Este último aspecto es crucial para entender los objetivos de ATAM, donde el objetivo no es predecir exactamente el comportamiento de un atributo de calidad. Esto será imposible en etapas tempranas en el escenario de diseño, porque todavía no se tiene suficiente información para hacer esta predicción. Es por ello que ATAM se centra en la identificación de riesgos.
Es sumamente importante en ATAM registrar cualquier riesgo, punto sensible y puntos de intercambio.
Los riesgos son decisiones arquitecturalmente importantes, que no han sido tomadas (ej. El equipo de arquitectura no ha decidido como distribuir el tiempo o no ha decidido si usará base de datos relacional o base de datos Orientada a Objetos) o decisiones que han sido tomadas pero las consecuencias no han sido entendidas a plenitud (ej. el equipo de arquitectura ha decidido incluir una capa de portabilidad en el sistema operativo, pero no están seguro que función usar para acceder dentro de la capa).

CAPÍTULO III: EVALUACIÓN DE LA ARQUITECTURA

Los puntos sensibles son parámetros en la arquitectura en los cuales la respuesta medible de algunos atributos de calidad son altamente correlacionados (ej. se puede determinar que la cantidad total de datos trasmitido en un sistema es altamente correlacional con la cantidad de datos que se transmiten por un canal de comunicación en específico).

Un punto de intercambio (tradeoff) es descubierto en la arquitectura cuando un parámetro de construcción arquitectural es un anfitrión para más de un punto sensible donde los puntos de calidad medibles son afectados indistintamente por cambio en el parámetro, (ej. si se aumenta la velocidad en el canal de comunicación mencionado anteriormente esto mejoraría en flujo de datos pero reduciría la confiabilidad, entonces la velocidad del canal es un punto de intercambio (tradeoff)).

La idea de caracterizar un atributo de calidad es un concepto clave sobre el cual se ha fundado ATAM así como:

- Los escenarios.
- Otro concepto sobre el cual se basa ATAM es el conocimiento sobre estilos arquitectónicos (ABAS).

ATAM es un método de análisis organizado alrededor de la idea que los estilos de arquitectura son determinantes de los atributos de calidad arquitectónicos (27). Los pasos para aplicar el método se explican continuación:

Presentación.

1. Presentar el ATAM: Se describe el método a los involucrados (Clientes representativos, Equipo de arquitectura)
2. Presentar la directriz del negocio: El jefe del proyecto describe las metas del negocio y en que lugares se debe centrar el esfuerzo, cual va ser la directriz que guiará la arquitectura (ej. alta disponibilidad y alta seguridad)
3. Presentar arquitectura: El arquitecto describe la arquitectura propuesta, centrándose en cómo va a estar direccionada hacia el negocio.

Investigación y análisis

4. Identificar estrategias arquitectónicas. Estas son identificadas por el arquitecto pero no son analizadas.
5. Generar árbol de utilidad de atributos de calidad. Los factores de calidad que comprenden la utilidad del sistema (rendimiento, disponibilidad, seguridad

modificabilidad) son capturados, especificados bajo un nivel de escenarios, comentados con estímulos, respuestas y priorizados.

6. Analizar estrategias arquitectónicas. Basada en los factores de mayor prioridad detectados en el paso 5, las estrategias arquitectónicas que guían esos factores son capturadas y analizadas (ej. una estrategia arquitectónica puede apuntar al entendimiento de las metas de rendimiento y esto puede ser subordinado a un análisis de rendimiento). Durante este paso son detectados los riesgos arquitectónicos, los puntos sensibles y los puntos de intercambio.

Pruebas

7. Tormenta de ideas y priorizar los escenarios. Basado en los escenarios obtenidos en el árbol de utilidad del paso 5, se agranda el número de escenarios obtenidos del grupo de involucrados. Este grupo de escenarios es priorizado mediante un proceso de votación que involucra a todo los interesados.
8. Analizar estrategia arquitectónica. Este paso repite el paso 6, pero aquí se resaltan los escenarios obtenidos en el paso 7.

Reporte

9. Presentar resultados.

3.6. Aplicación del método ATAM

En esta evaluación se analizarán como se comportan los siguientes atributos de calidad:

- Rendimiento.
- Mantenibilidad.

3.6.1. Presentación del negocio.

En el caso de la línea Planificación Empresarial y presupuestada para un mejor entendimiento del negocio se realizaron dos mapas de procesos, uno para la parte Presupuestada y otro para la parte de Empresarial.

Mapa de procesos del negocio de Planificación Empresarial:

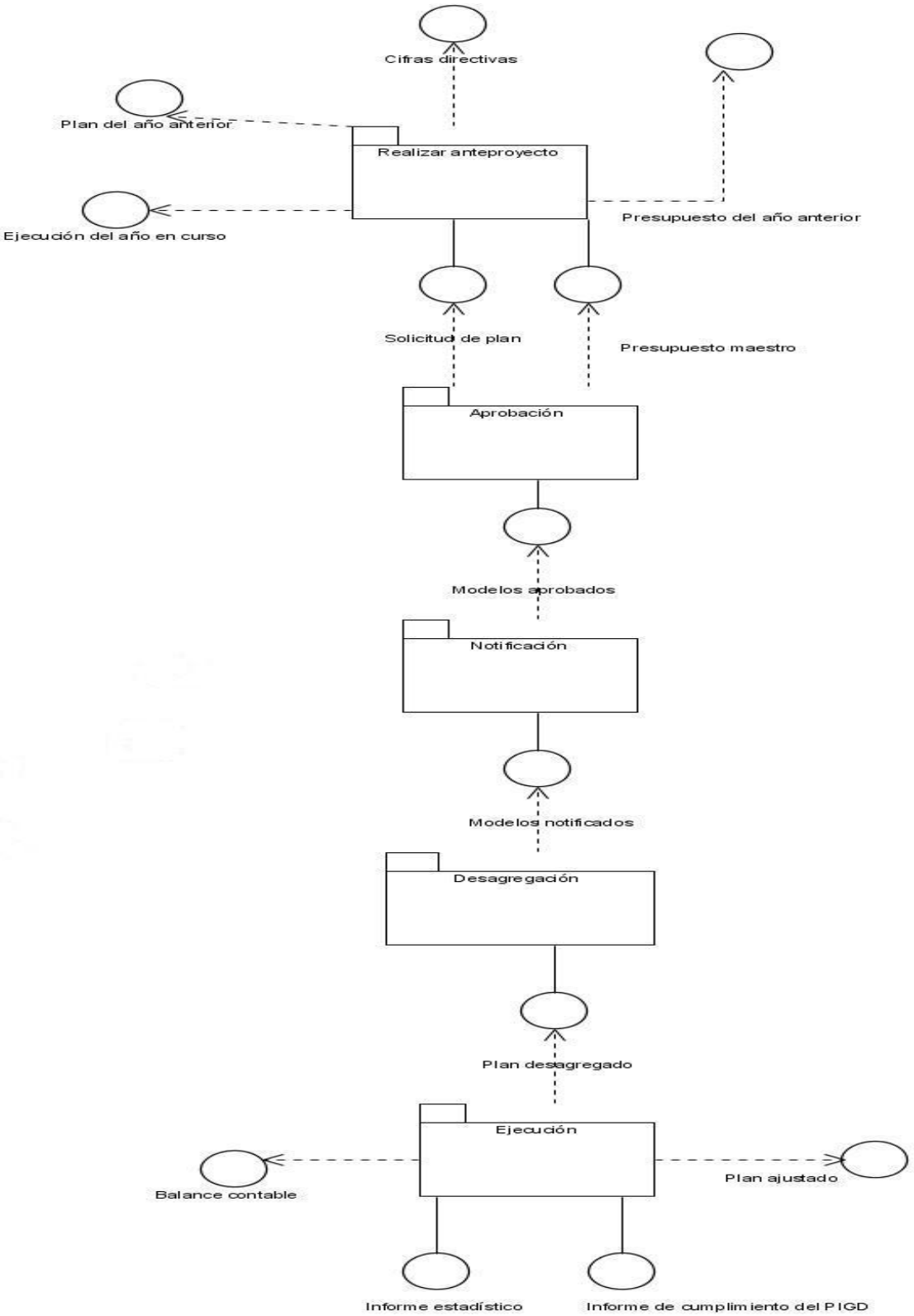


Figura 27: Mapa de Procesos de Negocio Planificación Empresarial

Descripción de Procesos de Negocio

Anteproyecto

CAPÍTULO III: EVALUACIÓN DE LA ARQUITECTURA

Las entidades empresariales se especializan y dependiendo de ese tipo, realizan los procesos del plan y el presupuesto empresarial, es decir del Presupuesto Maestro, que es el primer paso en la realización del anteproyecto. En este proceso el objetivo es que cada entidad realice su propuesta de plan.

Aprobación

Este proceso garantiza que dados los modelos generados por la etapa de Anteproyecto, una vez consolidados, sean elevados a los niveles superiores en orden jerárquico, para ser aprobados por las personas designadas para ello en los distintos niveles. Asegura la aprobación por los grupos empresariales y los Organismos de Administración Central del Estado, de los modelos empresariales establecidos por los ministerios, así como el Plan de Ingresos y Gastos en Divisas, el cual, es aprobado a nivel del MEP. En este proceso se incorporan las modificaciones que resulten de su análisis en los planes de las diferentes empresas.

Notificación

Una vez aprobado el plan para las empresas, los niveles superiores comienzan a notificar a sus subordinados el presupuesto aprobado. La notificación se realiza en orden inverso a la aprobación, iniciándose desde los niveles superiores hacia los inferiores; garantizando que lleguen las cifras aprobadas hasta las empresas. En esta etapa se notifican las restricciones impuestas a los planes de las diferentes entidades.

Desagregación

Desagregar por unidades, en columnas. Contando con los modelos de notificación correspondientes y el plan aprobado. Obteniéndose así el plan desagregado.

Ejecución

Este proceso tiene como objetivo analizar mensualmente la ejecución del presupuesto en cada empresa, para chequear que se esté cumpliendo lo planificado. En caso de existir desbalances en la ejecución del plan, se deben solicitar modificaciones a los niveles superiores, para ajustar nuevamente el plan si estas son aprobadas. Además las empresas deben emitir un informe estadístico a las Oficinas Territoriales de Estadística para que se almacene dicha información para su posterior análisis.

Mapa de procesos del negocio de Planificación Presupuestada:

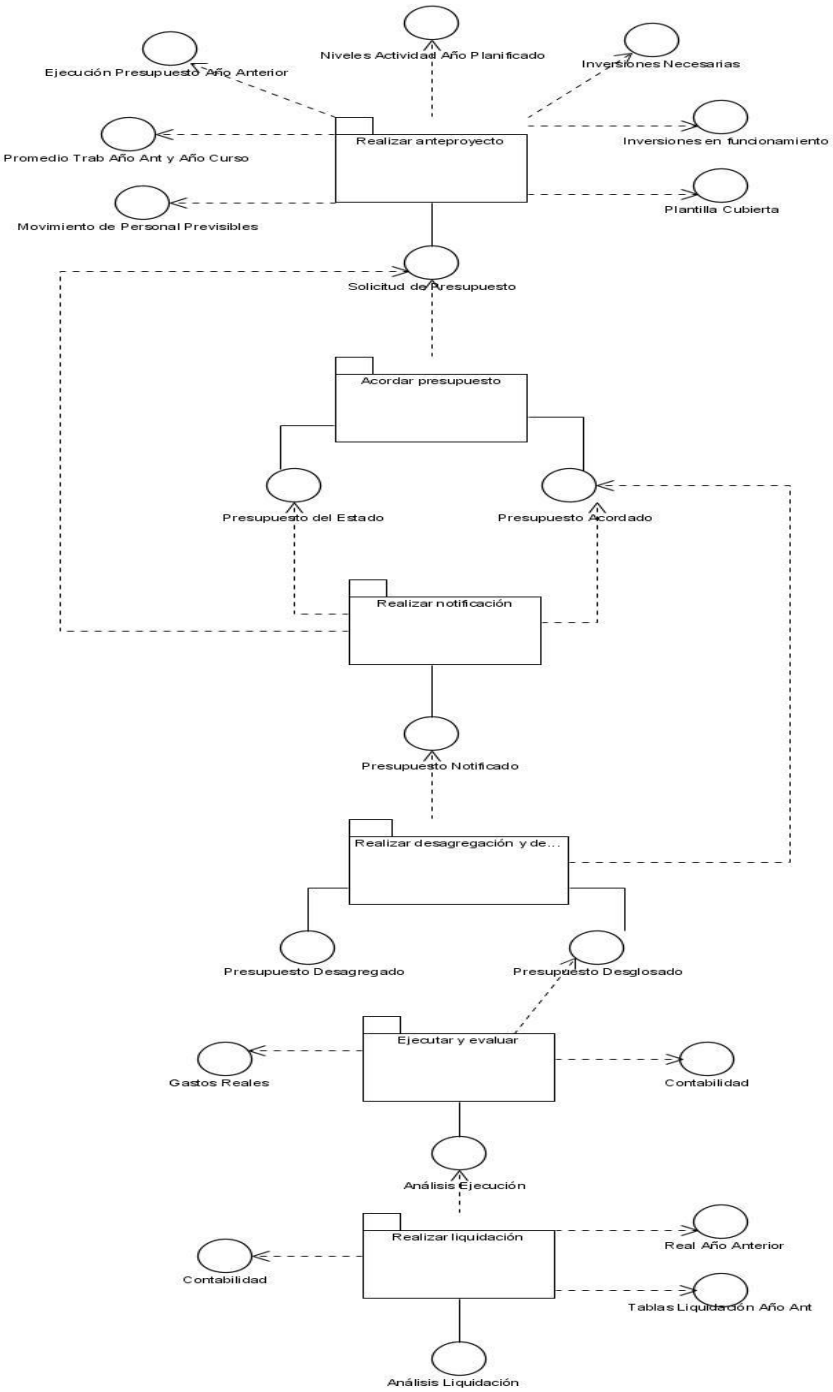


Figura 28: Mapa de Procesos de Negocio del Módulo de Planificación Presupuestaria

Descripción de los procesos del negocio

Realizar Anteproyecto

Este proceso comienza cuando la Dirección General de Presupuesto elabora las indicaciones para el Presupuesto del Estado del año siguiente, las cuales bajan hasta las unidades presupuestadas y estas elaboran su presupuesto, teniendo en cuenta las indicaciones antes mencionadas. Una vez que todas las organizaciones, unidades y direcciones presupuestadas

hayan realizado esta misma actividad se le presenta a la Dirección General de Presupuesto, la cual hace un análisis y ajusta las cifras que considere necesarias y ajustan el Presupuesto del Estado.

Acordar Presupuesto

Este proceso asegura la captación por el Ministerio de Finanzas y Precios y sus direcciones municipales y provinciales, de las cifras acordadas con las entidades presupuestadas una vez discutidos los anteproyectos presentados y consensuadas las cifras acordadas. En este proceso se incorporan las modificaciones que resulten de su análisis por el gobierno y la Asamblea Nacional.

Realizar Notificación

Una vez aprobado el Presupuesto del Estado por la Asamblea Nacional para el nuevo ejercicio fiscal, los especialistas de la Dirección General de Presupuesto preparan la Resolución donde se establecen los conceptos que constituyen límite de gasto y gasto de destino específico, una vez aprobada la misma se comienza a dar a conocer a los órganos y organismos del estado, organizaciones y asociaciones, otras entidades vinculadas al presupuesto, provincias, municipios, unidades presupuestadas y empresas, el presupuesto aprobado.

Realizar Desagregación y Desglose mensual

Este proceso tiene como objetivo realizar la desagregación por ramas, epígrafes, partidas y grupos presupuestarios y además desglosar por meses del ejercicio fiscal el presupuesto notificado.

Ejecutar y Evaluar

Este proceso tiene como objetivo analizar la ejecución del presupuesto, así como modificaciones presupuestarias, las mismas deben solicitarse por los Órganos de la Administración del Estado para cubrir los desbalances financieros que se presenten.

Realizar Liquidación

Este proceso tiene como objetivo analizar la ejecución del presupuesto al cierre del año, haciendo a cada nivel presupuestario una valoración completa final de la ejecución del presupuesto y del cumplimiento de las indicaciones específicas que rigieron para ese ciclo.

3.6.2. Descripción de la arquitectura.

El núcleo del marco de trabajo está desarrollado sobre la base de la plataforma LAPP(Linux-Apache-PostgreSQL-PHP), utilizando la versión de PHP 5.2.4 con la utilización de los frameworks; ZendFramework, Doctrine, ExtJS y ezComponent agregándole una extensión al ZendFramework (ZendExt) para darle solución a necesidades propias del negocio a desarrollar y del proceso del mismo, resolver potencialidades funcionales las cuales no se les pueden dar solución con los frameworks seleccionados, e incrementar la adaptabilidad a distintas núcleos de desarrollo incorporando un estilo híbrido basado en Capas y MVC.

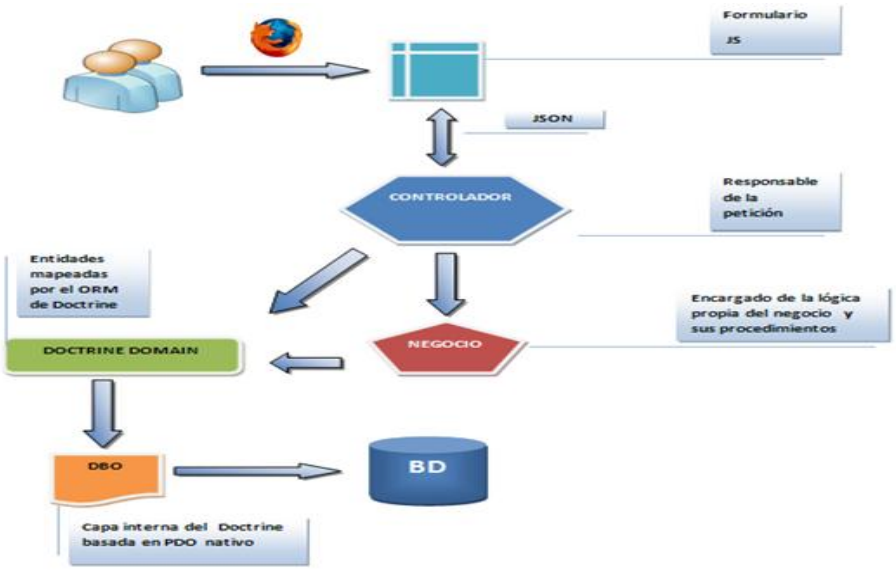


Figura 29: Estilo arquitectónico del marco de trabajo

En la figura 6 mostrada en el capítulo número 2 se presentó la interacción del núcleo del marco de trabajo, donde los aspectos arquitectónicos son respaldados por los componentes verticales del marco respondiendo además con las potencialidades funcionales de sistema como; **Caché, Conceptos globales, Validación, Trazas, Manejador de Excepciones,** entre otros. Se presenta un escenario simple dividido por capas (**Presentación**, **Negocio**, **Acceso a datos**), el cual interactúa con los componentes o subsistemas horizontales de la aplicación.

El estilo presentado en la ilustración está dirigido por la utilización de los frameworks ZendExt, ZendFramework y ExtJS el cual juega un papel esencial, donde desde el controlador se instancia la clase ZendView que es la encargada de edificar una interfaz gráfica, utilizando ficheros .js y .phtml.

En este estilo o capa, para las interacciones del cliente con el servidor se define:

- Peticiones HTTP utilizando la tecnología que brinda AJAX.
- Peticiones HTTP utilizando el componente iframe de HTML.

- Peticiones HTTP de inclusión de archivos de código .js y .css

Los formatos de intercambio definidos son:

JSON: para intercambio de datos entre componentes y demás elementos del servidor. Para archivos de multi-lenguaje

XML: para intercambio de datos que se almacenan archivos de configuración con formato XML.

Para el desarrollo de interfaces se trabaja en los siguientes aspectos:

- Nomenclatura para la definición de componentes.
- Definición mínima de uso para componentes del framework EXT.
- Validación de Componentes.
- Multilenguaje.
- Multitema.

Otro de los aspectos desarrollados en esta capa es la integración de interfaces a partir de una lógica determinada utilizando elementos que puedan utilizarse en una interfaz correspondiente a dicha lógica, desde otra totalmente independiente que necesite del servicio. Es similar a utilizar un servicio web, pero no a nivel de lógica sino a nivel de interfaces.

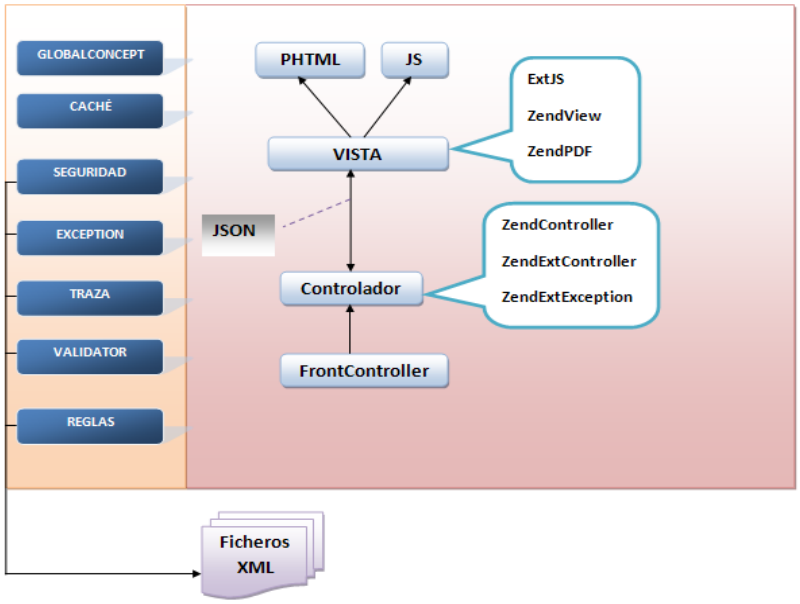


Figura 30: Escenario simple. Capa presentación

El estilo de la figura 31 donde el implementador define el negocio conjuntamente con sus procedimientos, se realiza instancias al dominio y se ejecutan las integraciones a nivel de lógica de negocio. Se establecen los servicios tanto internos como externos del componente para realizar integraciones entre los distintos componentes del subsistema.

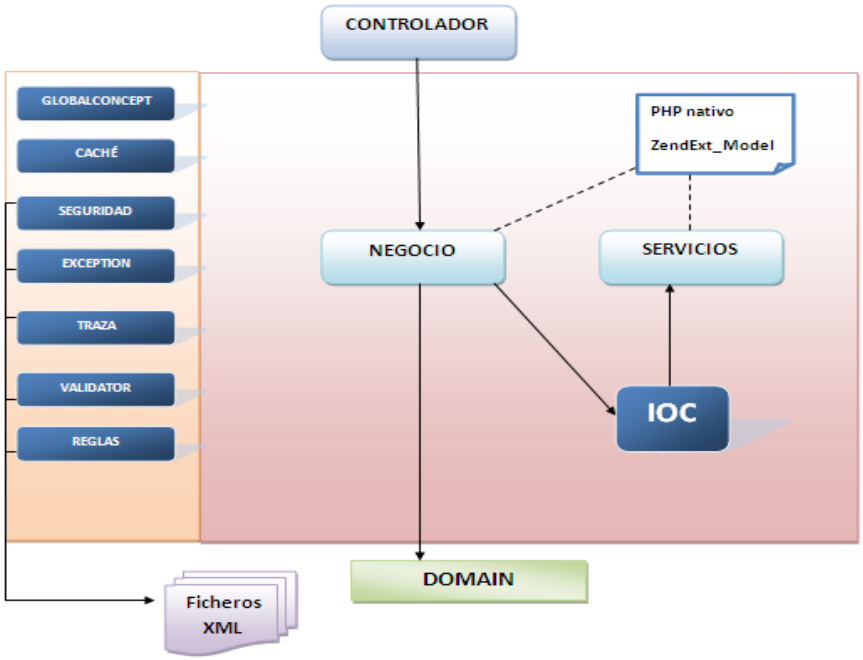


Figura 31: Escenario simple. Capa de negocio

La figura 31 presenta el estilo de acceso a datos Doctrine se encarga de la interacción del sistema con la base de datos mediante DBO (capa que utiliza Doctrine con PDO nativo) donde se exporta una base de datos existente a sus clases correspondientes y también a la inversa, es decir convertir clases (convenientemente creadas siguiendo las pautas del ORM) a tablas de una base de datos.

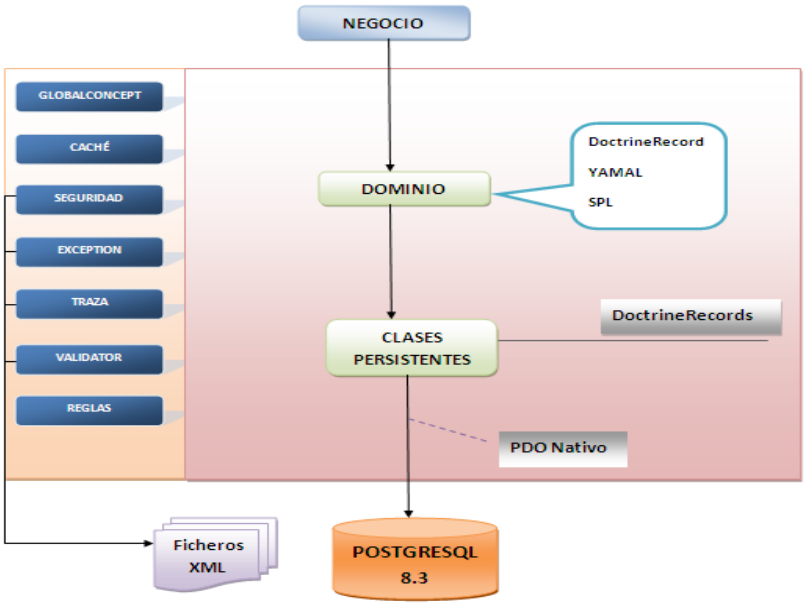


Figura 32: Escenario simple. Capa de acceso a datos

3.6.3. Implementación del MVC

Con la extensión del ZendFramework se define una redefinición del patrón MVC (Modelo-Vista-Controlador) parecido al que utiliza el ZendFramework, donde se exceptúa la interacción del modelo con la vista, y la interacción del controlador con el modelo es en un solo sentido (Controlador => Modelo).

Figura 33: MVC

3.6.4. Árbol de Utilidad de los atributos de calidad.

Tabla 8: Árbol de utilidad

Característica	Escenario
Rendimiento	Un usuario accede a cualquier interfaz de la aplicación y esta debe estar disponible en un período de 0.1 a 0.2 segundos
	Se solicita un recurso al sistema, que puede ser ligero (capacidad por debajo de 2 MB), medio (capacidad de 2 a 10 MB) o complejo (más de 10 MB de capacidad) en un servidor de 1 GB de memoria RAM y se recibe en un período de 0.1 a 0.7, de 0.8 a 2.0 y de 3.0 a 5.0 segundos respectivamente.
	Se intercambian datos con el sistema, ya sea mediante acciones de inserción, búsqueda o modificación, en un servidor de 1 GB de memoria RAM y se recibe la notificación de la acción realizada en un período de 0.1 a 0.7 segundos.
Mantenibilidad	Se modifican las características arquitectónicas del despliegue de la aplicación, el sistema deberá indicar las mismas cuando se realiza la próxima petición.

Después de haberse realizado un análisis se llegó a la conclusión que los principales atributos de calidad que van a afectar al proyecto son los descritos en el árbol de utilidad. Por eso se puede decir que el Subsistema Planificación Empresarial y Presupuestada va a estar afectado por estos atributos. Entonces la calidad del subsistema (**C**planif) se puede decir que es una función **f,** que es afectada por la calidad que tenga el desempeño de (**C**r), el rendimiento, y la (**C**m) Mantenibilidad.

Cplanif = f (Cr, Cm)

3.6.5. Análisis de los escenarios

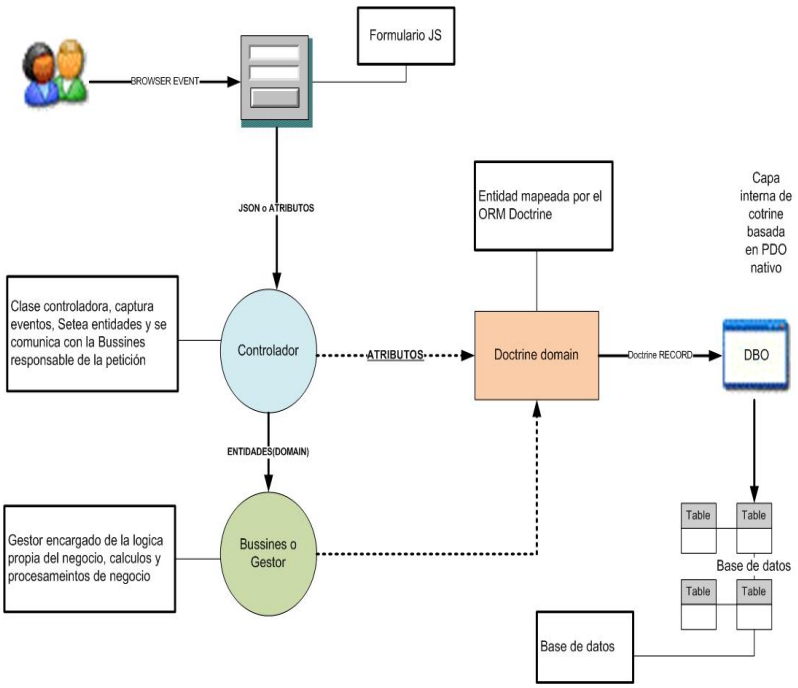
A continuación se presenta el análisis de uno de los escenarios arquitectónicos que es de interés dentro de la arquitectura.

Tabla 9: Escenario de prueba de atributos de calidad (Aplicación de ATAM)

Escenario:	Un usuario intercambia datos con el sistema.		
Atributo:	Rendimiento, Reusabilidad		
Ambiente:	En un servidor de 1 GB de memoria RAM en estado normal de explotación.		
Estímulo:	Se realizan operaciones en el sistema ya sea mediante acciones de inserción, búsqueda, modificación o eliminación.		
Respuesta:	Se recibe la notificación de la acción realizada en un período de 0.1 a 0.7 segundos.		
Decisiones de la arquitectura	*Riesgo*	*Puntos de sensibilidad*	*Puntos de desventaja*
DA1: Cuando se interactúa con una vista se dispara una clase controladora o sea, un js que llama a un controlador, este controlador no se comunica directamente con la clase que implementa el negocio si no que carga e instancia el objeto de doctrine. Este objeto carga de la base de datos desde donde va a las tablas por PDO. El Data Base Object a partir del PDO va a las tablas, carga la información	Disminuye el rendimiento.	Disminuye el rendimiento pero aumenta la reutilización.	Para cualquier acción tienen que ejecutarse por 6 componentes arquitectónicos.

construye el objeto y se lo devuelve al controlador que se comunica con la entidad y le pasa el objeto de dominio, se procesa la lógica en el gestor encargado de la lógica propia del negocio, cálculos y procesamiento del negocio. El controlador si tiene que persistir los datos vuelve a ir al doctrine le manda el objeto. El doctrine repite todo el mecanismo y regresa y concluye la aplicación. Una vez que concluye el controlador re-dibuja el js.			
Razonamiento:	El Framework a partir de cada tabla genera un objeto y toda la lógica asociada a modificar, eliminar, buscar y adicionar sin programar las clases. Solo se programa la lógica de negocio y el controlador que tiene. Aun cuando el rendimiento se ve afectado debido a que tiene que moverse por 6 componentes arquitectónicos es más productivo porque se genera mucho código automático que aumenta la reutilización. Esta evita tener que crear una cadena de conexión, la transacción, evita por cada tabla tener que crear una clase de atributos y además crear los métodos de adicionar, modificar, eliminar y buscar. Todo esto afecta el rendimiento. Por tanto cuando se haga una petición cuando se cargan los objetos no sean mas de 20 y se haga el procesamiento por lote, si se quiere mover 100 objetos se hacen peticiones de 20 en 20.		

Diagrama de la arquitectura:



Al realizar la evaluación arquitectónica se puede decir que los atributos de calidad que más prioridad se les dio por el riesgo que representan fueron Rendimiento y Mantenibilidad.

Se pudo observar que la utilización de ATAM es adecuada para la evaluación de este tipo de arquitecturas, ya que el método permite una evaluación temprana; es decir, no es necesario que toda la arquitectura esté definida para ser evaluada; el comportamiento observado en el alcance de la arquitectura evaluada se propaga a la arquitectura completa. Por esta razón se afirma que el comportamiento conseguido en la evaluación es válido para futuros componentes que se integren a la arquitectura de software, si ellos cumplen con los mismos patrones que los ya evaluados.

3.6.6. Decisiones

Una vez identificados los riesgos presentes en la arquitectura es responsabilidad de los tomadores de decisiones y Involucrados de la arquitectura, a partir de los riesgos encontrados efectuar la toma de decisiones.

Algunas de las decisiones tomadas para mitigar los riesgos en la arquitectura en cuanto al atributo de calidad rendimiento son:

- Rendimiento
    - Desarrollo de una auditoría al diseño arquitectónico del sistema.
    - Repetir las pruebas de evaluación de la arquitectura.

3.6.7. Evaluación de los resultados de la prueba

A partir de todos los elementos que se han reflejado, se puede afirmar que la solución propuesta presenta un rendimiento aceptable, y es altamente mantenible.

3.7. Conclusiones

El desarrollo de este capítulo, donde se le realizó una evaluación a la arquitectura aplicando el método ATAM, permitió descubrir los posibles puntos de riesgos de la arquitectura que se construía y las restricciones de la misma. A su vez permitió refinar y perfeccionar las decisiones arquitectónicas, contribuyó a la organización y a tener una mayor confianza en el producto que se construía, haciendo posible evitar la ocurrencia de costosos errores.

CONCLUSIONES

Durante el transcurso de la investigación y a través del trabajo realizado, se han llegado a las siguientes conclusiones generales:

Se realizó un estudio detallado de varios estilos arquitectónicos y patrones arquitectónicos y de diseño, lográndose un mejor entendimiento de ellos y un mayor aprovechamiento de sus ventajas en su aplicación a la arquitectura de la línea.

Se definió la arquitectura de sistema de la Línea Planificación Empresarial y Presupuestada, que sirvió de guía durante todo el proceso de desarrollo. Se realizaron todos los artefactos arquitectónicos definidos y de esta manera quedó implantada la arquitectura en el Subsistema.

Se realizó una evaluación de la arquitectura aplicando el método ATAM. A través del mismo se llegó a la conclusión que el atributo rendimiento, usado como caso de estudio fue aceptable.

RECOMENDACIONES

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

- Hacer un análisis más profundo siguiendo los pasos que propone el método ATAM en la arquitectura definida en la arquitectura definida.
- Hacer uso de la arquitectura del Subsistema de Planificación Empresarial y Presupuestada en futuras aplicaciones con esta línea de trabajo.

REFERENCIAS BIBLIOGRÁFICAS

1. Concepto e importancia de la planificación. [En línea] http://148.202.148.5/Cursos/Id204/Unidad_3/31.htm.

2. Arquitectura de la IP. [En línea] http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/arquitectura_soft.mspx..

3. ¿Arquitectura de Software? [En línea] http://siona.udea.edu.co/~aoviedo/Arquitectura%20de%20Software/Arquitectura%20de%20Software.htm#_Definiciones..

4. CAMACHO, ERIKA y NUÑEZ, GABRIEL. *ARQUITECTURAS DE SOFTWARE.* abril 2004.

5. G, HIDALGO. *MODELACIÓN DEL SUBSISTEMA DE CONTROL DE LA INFORMACION GEOESPACIAL EN LINEA.* 2008.

6. ALGUNOS TIPOS DE ARQUITECTURAS. *http://homepage.mac.com/imaz/iblog/C612772037/E20050907222635/Media/Algunos%20Tipos%20de%20Arquitecturas.pdf.* [En línea]

7. *Estilos arquitectónicos.* 2008.

8. ARQUITECTURA Modelo/Vista/Controlador. [En línea] http://www.cica.es/formacion/JavaTut/Intro/tabla.html.

9. Reynoso, Carlos y Kiccillof, Nicolás. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2008.

10. Larman, Graig. *UML y Patrones. Introducción al analisis y diseño orientado a objetos.* 2008.

11. GAMMA, ERICH. *Design Patterns: Elements of.* 1995.

12. E, Mendoza M. L. SISTEMAS DE INFORMACIÓN II TEORÍA. [En línea] 2008. http://prof.usb.ve/lmendoza/Documentos/PS-6116/Teor%EDa%20PS6116%20Reingenier%EDa.pdf.

13. Pimentel, Luis Alberto y Perez, Yosev. *ArBaWeb: ARQUITECTURA BASE SOBRE LA WEB.* 2008.

14. El ataque de los frameworks. [En línea] http://www.webtaller.com/maletin/articulos/el-ataque-de-los-frameworks.php..

15. Sistema de gestión de base de datos. [En línea] http://www.igac.gov.co:8080/igac_web/UserFiles/File/ciaf/TutorialSIG_2005_26_02/paginas/ctr_sistemasdegestiondebasededatos.htm. .

16. Collins-Sussman, B, Fitzpatrick, B y Pilato, W.B. Control de versiones con Subversion. [En línea] 2008. http://svnbook.red-bean.com/nightly/es/svn-book.pdf.

REFERENCIAS BIBLIOGRÁFICAS

17. Línea Base. [En línea] 2009. http://10.12.171.3/svn/erp/\ERP\Ingenieria\Arquitectura\Desarrollo\Arquitectura sistema. .

18. SOLA, ELIANYS HURTADO. Ayuda al usuario del caso de estudio en el nuevo marco de trabajo del ERP. [En línea] http://10.12.171.3/svn/erp/\ERP\Ingenieria\Arquitectura\Desarrollo\Arquitectura sistema. .

19. Campis, Virgen Damaris Quevedo. Documento Especificación de la Arquitectura de Sistema e Integración ERP Cuba. [En línea] 2009. http://10.12.171.3/svn/erp/\ERP\Ingenieria\Arquitectura\Desarrollo\Arquitectura sistema. .

20. REYES, ALBERTO HIDALGO. *MODELACIÓN DE ARQUITECTURA PARA APLICACIONES EMPRESARIALES EN PHP.* 2008.

21. Normas y estándares de Codificación del ERP. [En línea] http://10.12.171.3/svn/erp/\ERP\Ingenieria\Arquitectura\Desarrollo\Arquitectura sistema. .

22. Estándar de diseño de interfaces para las aplicaciones de gestión. [En línea] http://10.12.171.3/svn/erp/\ERP\Ingenieria\Arquitectura\Desarrollo\Arquitectura sistema..

23. Plantillas: Plan de Iteración. [En línea] http://synergix.wordpress.com/2008/07/18/plantillas-plan-de-iteracion/.

24. Plan de Iteración. [En línea] http://merinde.rinde.gob.ve/index.php?option=com_content&task=view&id=106&Itemid=301. .

25. CARRASCO PUEBLA, YOAN ARLET y CÉSPEDES VEGA, ANISLEYDIS. *Procedimiento para la Evaluación de Arquitecturas de Software basadas en Componentes.* 2008.

26. GÓMEZ, O. S. *Evaluando Arquitecturas de Software. Parte 1.*

27. KAZMAN, R y KLEIN, M. *ATAM: Method for Architecture Evaluation.* 2000.

BIBLIOGRAFÍA.

Algunos tipos de arquitecturas. 2008, nº disponible en: http://homepage.mac.com/imaz/iblog/c612772037/e20050907222635/media/algunos%20tipos%20de%20arquitecturas.pdf.

¿Arquitectura de software? disponible en: http://siona.udea.edu.co/~aoviedo/arquitectura%20de%20software/arquitectura%20de%20software.htm#_definiciones.

Arquitectura de la ip disponible en: http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/arquitectura_soft.mspx.

Arquitectura, l. d. Normas y estándares de codificación del erp disponible en: http://10.12.171.3/svn/erp/\erp\ingenieria\arquitectura\desarrollo\arquitectura sistema.

Alberto Hidalgo Reyes, j. v. s. Modelación de arquitectura para aplicaciones empresariales en php. 2008.

Collins-Sussman, b. f., w.b. pilato, m.c. Control de versiones con subversion. 2008.

Campins, v. d. q. Documento especificación de la arquitectura de sistema e integración erp cuba disponible en: http://10.12.171.3/svn/erp/\erp\ingenieria\arquitectura\desarrollo\arquitectura sistema.

Erika Camacho, F. C., Gabriel Nuñez. arquitecturas de software. abril – 2004.

Elianys Hurtado Sola, y. m. b. Ayuda al usuario del caso de estudio en el nuevo marco de trabajo del erp disponible en: http://10.12.171.3/svn/erp/\erp\ingenieria\arquitectura\desarrollo\arquitectura sistema.

E, M. M. l. Sistemas de información ii teoría disponible en: http://prof.usb.ve/lmendoza/documentos/ps6116/teor%eda%20ps6116%20reingenier%eda.pdf

Estándar de diseño de interfaces para las aplicaciones de gestión. disponible en: http://10.12.171.3/svn/erp/\erp\ingenieria\arquitectura\desarrollo\arquitectura sistema

El ataque de los frameworks disponible en: http://www.webtaller.com/maletin/articulos/el-ataque-de-los-frameworks.php.

BIBLIOGRAFÍA

Estilos y patrones en la estrategia de arquitectura de microsoft (versión 1.0 – marzo de 2004) CARLOS REYNOSO – NICOLÁS KICCILLOF. 2008.

Estilos arquitectónicos. 2008.

HIDALGO.G. Modelación del subsistema de control de la informacion geoespacial en linea. 2008, nº

Gómez, O. S. Evaluando arquitecturas de software. parte 1. Panorama general. 2007, nº

Gustavo andrés brey, G. E., Nicolas Passerini y Juan Arias. Arquitectura de Proyectos de iterativos. Evaluación de arquitecturas. 2005, nº

González, d. b. línea base. 2009, disponible en: http://10.12.171.3/svn/erp/\erp\ingenieria\arquitectura\desarrollo\arquitectura sistema.

Pimentel, l. y. p. i. arbaweb: arquitectura base sobre la web. 2008.

Plantillas: plan de iteración disponible en: http://synergix.wordpress.com/2008/07/18/plantillas-plan-de-iteracion/.

Plan de iteración disponible en: http://merinde.rinde.gob.ve/index.php?option=com_content&task=view&id=106&itemid=301.

Roger. pressman. ingeniería de software.un enfoque práctico.vol 5 quinta edición.

Kazman, r., m. klein, et al. atam: method for architecture evaluation. 2000.

Sistema de gestión de base de datos disponible en: http://www.igac.gov.co:8080/igac_web/userfiles/file/ciaf/tutorialsig_2005_26_02/paginas/ctr_sistemasdegestiondebasededatos.htm.

Yoan Arlet Carrasco Puebla, E. C. G., Anisleydis céspedes vega. procedimiento para la evaluación de arquitecturas de software basadas en componentes. 2008, Nº

GLOSARIO DE TÉRMINOS

PHP: Es un acrónimo recursivo que significa ***P****HP* ***H****ypertext* ***P****re-processor* (inicialmente PHP Tools, o, *P*ersonal *H*ome *P*age Tools). Es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas.

Servicio web: (en inglés *Web service*) Es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos.

Servicio: Una función sin estado (Existen servicios asíncronos en los que una solicitud a un servicio crea, por ejemplo, un archivo, y en una segunda solicitud se obtiene ese archivo), auto-contenida, que acepta una(s) llamada(s) y devuelve una(s) respuesta(s) mediante una interfaz bien definida. Los servicios pueden también ejecutar unidades discretas de trabajo como serían editar y procesar una transacción. Los servicios no dependen del estado de otras funciones o procesos. *La tecnología concreta utilizada para prestar el servicio no es parte de esta definición*.

Web: (" World Wide Web ") o Red Global Mundial es un sistema de documentos de hipertexto y/o hipermedios enlazados y accesibles a través de Internet. Con un navegador Web, un usuario visualiza páginas web que pueden contener texto, imágenes, vídeos u otros contenidos multimedia, y navega a través de ellas usando hiperenlaces.

XML: Sigla en inglés de *E**x**tensible* ***M****arkup* ***L****anguage* («lenguaje de marcas extensible»), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

GLOSARIO DE TÉRMINOS

JSON: Acrónimo de "**J**ava**S**cript **O**bject **N**otation", es un formato ligero para el intercambio de datos. **JSON** es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML

JavaScript: Es un lenguaje de programación interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas web.

Herramientas CASE: (***C****omputer* ***A****ided* ***S****oftware* ***E****ngineering*, Ingeniería de Software Asistida por Ordenador) Son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

HTTP: (HyperText Transfer Protocol): Protocolo de transferencia de hipertexto, es el protocolo usado en cada transacción de la Web (WWW). El hipertexto es el contenido de las páginas web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceso a una página y la respuesta con el contenido.

Abstracción: Consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan. En programación, el término se refiere al énfasis en el "¿qué hace?" más que en el "¿cómo lo hace?" (Característica de caja negra). El común denominador en la evolución de los lenguajes de programación, desde los clásicos o imperativos hasta los orientados a objetos, ha sido el nivel de abstracción del que cada uno de ellos hace uso.

IoC: Tradicionalmente el programador especifica la secuencia de decisiones y procedimientos que pueden darse durante el ciclo de vida de un programa mediante llamadas a funciones. En su lugar, en la inversión de control se especifican respuestas deseadas a sucesos o solicitudes de datos concretas, dejando que algún tipo de entidad o arquitectura externa lleve a cabo las acciones de control que se requieran en el orden necesario y para el conjunto de sucesos que tengan que ocurrir. El flujo habitual se da cuando es el código del usuario quien invoca a un procedimiento de una librería. La inversión de control sucede cuando es la librería la que invoca el código del usuario. Típicamente sucede cuando la librería es la que implementa las estructuras de alto nivel y es el código del usuario el que implementa las tareas de bajo nivel. La utilización de interfaces y la aparición de los frameworks ha popularizado este término. De hecho es el concepto central del Framework de Spring, ya que implementa un "Contenedor" que se encarga de gestionar las instancias (así como sus creaciones y destrucciones) de los objetos del usuario. Por tanto las aplicaciones

que utilicen el framework de Spring (no Spring propiamente dicho) utilizarán Inversión de Control.

ANEXOS

Anexo 1 Modelo de datos

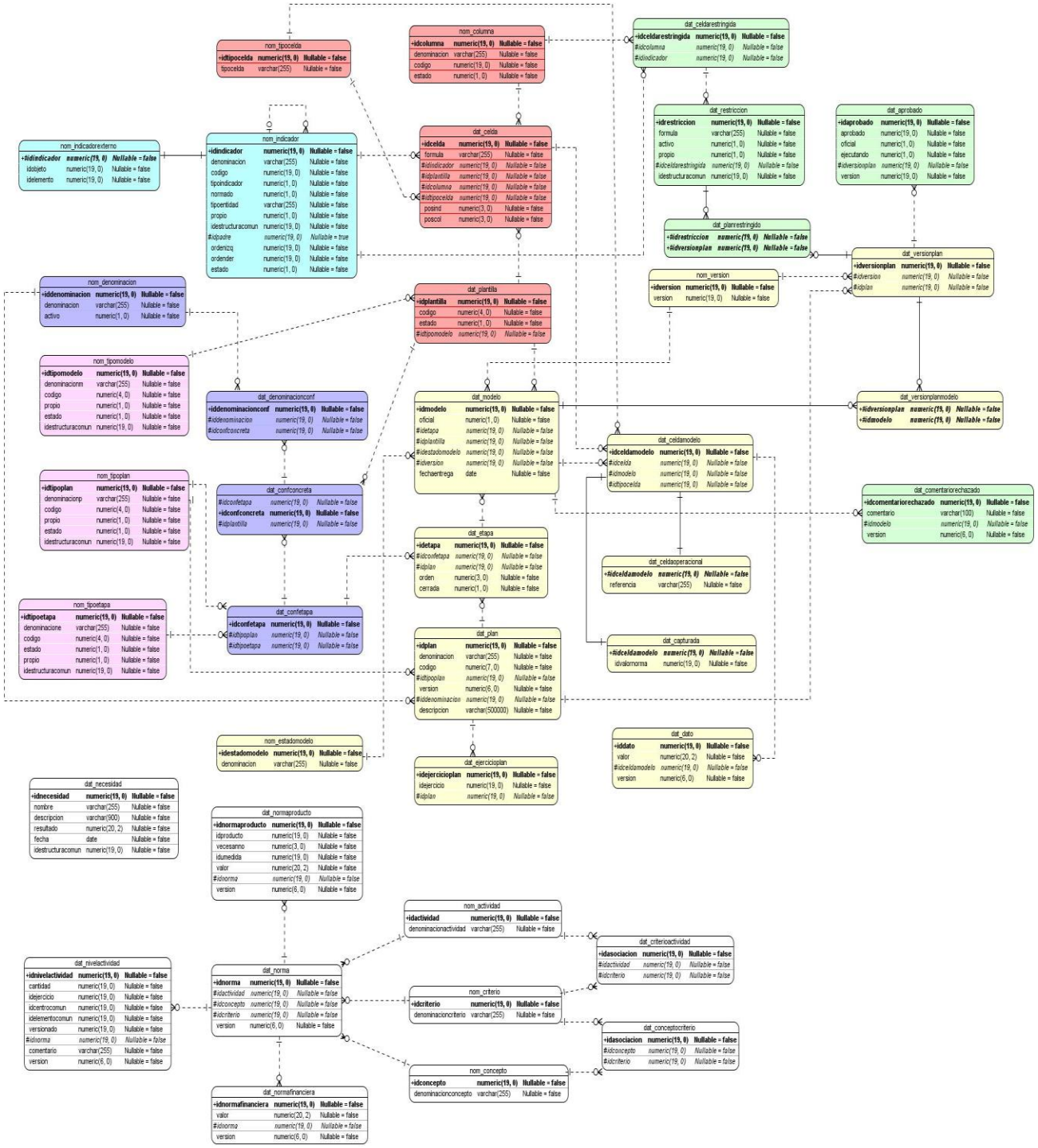


Figura 34: Modelo de datos de la línea Planificación Empresarial y Presupuestada

ANEXOS

Anexo 2 Diccionario de datos.

Tabla 10 Atributo

Nombre de la entidad	Atributo					
Descripción de la entidad	Es una característica que define el comportamiento de un indicador.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas
Denominación	El nombre con que se identificará el atributo.	Cadena de caracteres.	No.	Si.	Cadena de caracteres	No procede

Tabla 11 Indicador

Nombre de la entidad	Indicador					
Descripción de la entidad	Es un concepto medible.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas
Denominación	El nombre con que se identificará el atributo.	Cadena de caracteres.	No.	Si.	Cadena de caracteres.	No procede

Tabla 12 Restricción

Nombre de la entidad	Restricción					
Descripción de la entidad	Límites que serán evaluados y validados durante la planificación, restringiendo los diferentes valores a tomar por los indicadores para un alias determinado.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas

Denominación	Nombre que identificará cada restricción.	String	No.	No.	Cadena de caracteres	No procede

Tabla 13 Límite

Nombre de la entidad	Límite					
Descripción de la entidad	Tipo de restricción que está enmarcada entre un valor mínimo y uno máximo.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas
Valor mínimo	Menor valor que puede tomar una restricción de este tipo.	int	No.	Si.	Números	Letras. Caracteres extraños.
Valor máximo	Mayor valor que puede tomar una restricción de este tipo.	int	No.	Si.	Números.	Letras. Caracteres extraños.

Tabla 14 Fórmula

Nombre de la entidad	Fórmula					
Descripción de la entidad	Forma de calcular un valor mediante números, operadores.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no

						válidas
Fórmula	Fórmula a elaborar.	String.	No.	Si.	Todas.	No procede

Tabla 15 Comentario

Nombre de la entidad	Comentario					
Descripción de la entidad	Texto de sugerencia o señalamiento realizado al revisar un modelo, que puede estar relacionado, o no, con una restricción.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas
Texto	Texto con las sugerencias, señalamientos.	String.	No.	Si.	Todas.	No procede.

Tabla 16 Celda editada

Nombre de la entidad	Celda editada					
Descripción de la entidad	Es la celda donde se le da la posibilidad al usuario de entrar él mismo los datos.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas

Tabla 17 Celda calculada

Nombre de la entidad	Celda calculada					
Descripción de la entidad	Es la celda en la que se hace referencia a una expresión matemática que utiliza, para determinar su valor, los valores que tomen otras celdas de la misma plantilla, una vez construido un modelo que la utilice.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no

						válidas

Tabla 18 Celda capturada

Nombre de la entidad	Celda capturada					
Descripción de la entidad	Es la celda en la que se hace referencia a una expresión matemática que utiliza, para determinar su valor, los valores que tomen otras celdas de diferentes plantillas, una vez construidos los modelos que las utilicen.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas

Tabla 19 Plantilla fórmula

Nombre de la entidad	Plantilla fórmula					
Descripción de la entidad	Representa la acción de la búsqueda del valor de la celda capturada fuera de la plantilla.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas
Plantilla fórmula	Representa la expresión a la que hace referencia la celda.	Cadena de caracteres	No.	No.	Todas	Ninguna

Tabla 20 Plan

Nombre de la entidad	Plan					
Descripción de la entidad	Representa un proyecto que elabora toda entidad para ejecutar su contabilidad en un período de tiempo, que se elabora anticipadamente con la intención de guiar su economía.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no

						válidas
Denominación	Nombre que se le da a un plan.	Cadena de caracteres.	No.	No.		
Ejecución	Determina si un plan está o no en ejecución.	Booleano.	No.	No.		
Rango	Determina el rango de ejercicios que abarcará el plan.	Fecha	No.	No.	2009-2010 2009	

Tabla 21 Ejercicio

Nombre de la entidad	Ejercicio					
Descripción de la entidad	Período de tiempo, normalmente un año, en que una institución o empresa dividen su actividad económica y durante el cual rige una ley de presupuestos.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas
Denominación	Nombre que identifica un período	Cadena de caracteres	No	No	Todas	Ninguna

Tabla 22 Período

Nombre de la entidad	Período					
Descripción de la entidad	Espacio de tiempo en los que se divide un ejercicio.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas
Denominación	Nombre que identifica un período	Cadena de caracteres	No	No	Todas	Ninguna

ANEXOS

Tabla 23 Configuración del Plan

Nombre de la entidad	Configuración del Plan					
Descripción de la entidad	Es una relación que se establece entre un Plan-Etapa-Modelo-Plantilla, para facilitar la planificación.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas
Denominación	Nombre que identifica una Configuración del Plan.	Cadena de caracteres	No	Si	Todas	Ninguna
Tipo de Plan	Nombre del Plan.	Es un dato que se selecciona	No	No	Todas	Ninguna
Etapas	Nombre de las etapas, por las que atraviesa el plan.	Es un dato que se selecciona	No	No	Todas	Ninguna
Modelo	Nombre de las modelos, con los que cuenta el plan.	Es un dato que se selecciona	No	No	Todas	Ninguna
Plantilla	Nombre de las Plantillas del plan	Es un dato que se selecciona	No	No	Todas	Ninguna

Tabla 24 Etapa

Nombre de la entidad	Etapa					
Descripción de la entidad	Es un momento por el que atraviesa la planificación.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no

						válidas
Denominación	Nombre que identifica una Etapa.	Cadena de caracteres	No	No	Todas	Ninguna

Tabla 25 Modelo

Nombre de la entidad	Modelo					
Descripción de la entidad	Tabla con datos que responde a una plantilla específica.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas
Versión	Número que se utiliza para registrar los cambios.	Números enteros	No	No	Números enteros	Todas las demás
Fecha	Fecha en que se cambia de estado un modelo.	Fecha	No	No	Formatos de fecha	Todas las demás
Estado	Fases por las que transita un modelo.	Letras	No	No	Cadena de letras	Todas las demás
Oficial	Es el modelo final.	Booleano	Si	Si	Todas	Ninguna

Tabla 26 Valor celda

Nombre de la entidad	Valor celda					
Descripción de la entidad	Es la cifra que tiene la celda. Puede ser introducida por el usuario, o calcularse automáticamente, en dependencia del tipo de celda en la que se encuentre.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas
Valor	Cifra que tiene la celda.	Número	No	Si	Números	Todas las demás.

ANEXOS

Tabla 27 Plantilla

Nombre de la entidad	Plantilla					
Descripción de la entidad	Es una tabla vacía que sirve de base para hacer un modelo.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas
Propio de la entidad	Una plantilla puede ser elaborada por la entidad u orientada por sus superiores.	Booleano	Si	Si	Todas	Ninguna
Estado	Fases por las que transita una plantilla.	Letras	No	Si	Cadena de letras	Todas las demás
Código	Identificador de la plantilla.	Números enteros	No	Si	Número de 4 cifras	Todas las demás.

Tabla 28 Celda

Nombre de la entidad	Celda					
Descripción de la entidad.	Es un campo de la plantilla que se configura en dependencia de sus características. Pueden clasificarse en Calculada, Capturada y Editada.					
Nombre del atributo	Descripción	Tipo	¿Puede ser nulo?	¿Es único?	Restricciones	
					Clase válidas	Clases no válidas
Bloqueada	Es una característica que se le atribuye a una celda a la que no se le pueden introducir datos.	Booleano	No	Si	1 o 2	Todas las demás.

ANEXOS

Anexo 3 Plan de iteración de los componentes

ANEXOS

lombre de tarea	Línea	Área funcional	Duration	Start	Finish	% Complete
Desarrollo del Subsistema Planificación	**Planificación**		##########	**Fri 25/04/08**	**Wed 27/05/09**	**83%**
Diseño	**Planificación**	**'roducción**	##########	##########	**Tue 17/03/09**	**100%**
Construcción	**Planificación**	**'roducción**	##########	**Fri 25/04/08**	**Sat 16/05/09**	**86%**
Componentes	**Planificación**	**'roducción**	##########	**Fri 25/04/08**	**Sat 16/05/09**	**95%**
Nomencladores	**Planificación**	**'roducción**	**40,44 days?**	**Fri 13/02/09**	**Thu 09/04/09**	**100%**
Entrega a calidad del componente Nomencladores	Planificación	Producción	0,02 days?	Sat 28/02/09	Sat 28/02/09	100%
Indicadores	**Planificación**	**'roducción**	**69,56 days?**	##########	**Fri 15/05/09**	**80%**
Entrega a calidad del componente Indicador	Planificación	Producción	0 days?	Fri 15/05/09	Fri 15/05/09	0%
Plantilla	**Planificación**	**'roducción**	**62,94 days?**	**Fri 13/02/09**	**Sat 09/05/09**	**93%**
Entrega a calidad del componente Plantilla	Planificación	Producción	0,02 days?	Fri 13/03/09	Fri 13/03/09	100%
Configuración de Plan	**Planificación**	**'roducción**	**24,44 days?**	##########	**Mon 30/03/09**	**100%**
Entrega a calidad del componente Configuración de Plan	Planificación	Producción	10,4 days?	Fri 27/03/09	Fri 10/04/09	100%
Construcción de Plan	**Planificación**	**'roducción**	**66,22 days?**	##########	**Fri 08/05/09**	**98%**
Entrega a calidad del componente Construcción de plan	Planificación	Producción	0,5 days?	##########	Mon 20/04/09	100%
Realización de Plan	**Planificación**	**'roducción**	##########	**Fri 25/04/08**	**Mon 11/05/09**	**99%**
Entrega a calidad del componente Realización de Plan	Planificación	Producción	0,12 days?	##########	Mon 11/05/09	0%
Cálculo de Necesidades	**Planificación**	**'roducción**	##########	##########	**Mon 04/05/09**	**92%**
Entrega a calidad del componente Cáculo de Necesidades	Planificación	Producción	0,03 days?	##########	Mon 27/04/09	100%
Procesador matemático	**Planificación**	**'roducción**	**77,42 days?**	##########	**Wed 13/05/09**	**97%**
Entrega a calidad del componente Procesador matemático	Planificación	Producción	0,2 days?	Fri 08/05/09	Fri 08/05/09	100%
Consultar	**Planificación**	**'roducción**	**31 days?**	##########	**Wed 13/05/09**	**93%**
Entrega a calidad del componente Construcción de plan	Planificación	Producción	0,11 days?	##########	Wed 13/05/09	0%
Otras Tareas	**Planificación**	**'roducción**	**42,56 days?**	**Fri 20/03/09**	**Sat 16/05/09**	**52%**
Recuperaciones de Planificación	**Planificación**	**'roducción**	**7 days**	##########	**Thu 14/05/09**	**0%**
Terminación de Recuperaciones del Subsistema Planificación	Planificación	Producción	0,45 days?	Thu 14/05/09	Thu 14/05/09	0%
Elaboración de los Manuales de Usuario de Subsistema Planifi	**Planificación**	**'roducción**	**35,28 days?**	##########	**Fri 15/05/09**	**74%**
Elaboración de Ayuda del subsistema de Subsistema Planifica	**Planificación**	**'roducción**	**31,5 days?**	##########	**Mon 11/05/09**	**0%**
Integración y Prueba	**Planificación**	**'roducción**	##########	**Fri 25/04/08**	**Wed 27/05/09**	**36%**

Figura 35: Plan de iteración de los componentes

Anexo 4 Ejemplo de la iteración de un componente

Nombre de tarea	Línea	Área funcional	Duration	Start	Finish	% omplet	Resource Names
Implementación del componente Plantill	**Planificación**	**Producción**	**20 days?**	**Tue 24/02/09**	**Tue 24/03/09**	**100%**	**berto Ruiz;Ariadna Rendó**
Gestionar plantilla	**Planificación**	**Producción**	**10,83 days?**	**Fri 27/02/09**	**Fri 13/03/09**	**100%**	**berto Ruiz;Ariadna Rendó**
Adicionar plantilla	Planificación	Producción	4,72 days?	Fri 27/02/09	Wed 11/03/09	100%	z;Alberto Ruiz;Ariadna Rendó
Modificar plantilla	Planificación	Producción	5,67 days	Fri 06/03/09	Fri 13/03/09	100%	z;Alberto Ruiz;Ariadna Rendó
Eliminar plantilla	Planificación	Producción	2,39 days?	Wed 11/03/09	Fri 13/03/09	100%	z;Alberto Ruiz;Ariadna Rendó
Activar plantilla	Planificación	Producción	2,39 days?	Wed 11/03/09	Fri 13/03/09	100%	z;Alberto Ruiz;Ariadna Rendó
Desactivar plantilla	Planificación	Producción	2,39 days?	Wed 11/03/09	Fri 13/03/09	100%	z;Alberto Ruiz;Ariadna Rendó
Consultar plantilla	Planificación	Producción	2,39 days?	Wed 11/03/09	Fri 13/03/09	100%	z;Alberto Ruiz;Ariadna Rendó
Gestionar columnas	**Planificación**	**Producción**	**0,17 days?**	**Wed 25/02/09**	**Wed 25/02/09**	**100%**	**berto Ruiz;Ariadna Rendó**
Adicionar columna	Planificación	Producción	0,17 days?	Wed 25/02/09	Wed 25/02/09	100%	z;Alberto Ruiz;Ariadna Rendó
Eliminar columna	Planificación	Producción	0,06 days?	Wed 25/02/09	Wed 25/02/09	100%	z;Alberto Ruiz;Ariadna Rendó
Gestionar filas	**Planificación**	**Producción**	**1 day?**	**Thu 26/02/09**	**Fri 27/02/09**	**100%**	**berto Ruiz;Ariadna Rendó**
Adicionar fila	Planificación	Producción	0,17 days?	Thu 26/02/09	Thu 26/02/09	100%	z;Alberto Ruiz;Ariadna Rendó
Eliminar fila	Planificación	Producción	0,06 days?	Fri 27/02/09	Fri 27/02/09	100%	z;Alberto Ruiz;Ariadna Rendó
Gestionar columnas independientes	**Planificación**	**Producción**	**0,19 days?**	**Tue 24/02/09**	**Tue 24/02/09**	**100%**	**berto Ruiz;Ariadna Rendó**
Agregar columnas	Planificación	Producción	0,08 days?	Tue 24/02/09	Tue 24/02/09	100%	z;Alberto Ruiz;Ariadna Rendó
Modificar columnas	Planificación	Producción	0,08 days	Tue 24/02/09	Tue 24/02/09	100%	z;Alberto Ruiz;Ariadna Rendó
Desactivar columnas	Planificación	Producción	0,03 days	Tue 24/02/09	Tue 24/02/09	100%	z;Alberto Ruiz;Ariadna Rendó
Activar columnas	Planificación	Producción	0,03 days	Tue 24/02/09	Tue 24/02/09	100%	z;Alberto Ruiz;Ariadna Rendó
Consultar columnas	Planificación	Producción	0 days	Tue 24/02/09	Tue 24/02/09	100%	z;Alberto Ruiz;Ariadna Rendó
Configurar celdas	**Planificación**	**Producción**	**9,22 days?**	**Wed 11/03/09**	**Tue 24/03/09**	**100%**	**berto Ruiz;Ariadna Rendó**
Configurar celda calculada	Planificación	Producción	2,39 days?	Wed 11/03/09	Tue 24/03/09	100%	z;Alberto Ruiz;Ariadna Rendó
Integración del componente Plantilla	**Planificación**	**Producción**	**53,56 days?**	**Thu 26/02/09**	**Sat 09/05/09**	**60%**	**Rodríquez;Ariadna Rendó**
Elaboración de Diseños de Casos de Prueba (	Planificación	Producción	2,39 days?	Wed 11/03/09	Fri 13/03/09	100%	nia Macias;Aliuska Valenzue
Pruebas Internas del componente Plantilla	Planificación	Producción	9,89 days?	Fri 27/02/09	Fri 13/03/09	100%	Zenia Macias;Alianet Puente

Figura 36: Ejemplo de utilización del Plan de iteración de los componentes

ANEXOS

Anexo 5 Priorización de los componentes

C = Componentes

R = Requisitos

CI = Complejidad de implementación del requisito (valor entre 1 y 10)

CIT = Complejidad de integración del requisito (valor entre 1 y 10)

E = Entidades

CE = Cantidad de entidades

GN = Gestores del negocio

CGN = Cantidad de Gestores del negocio

CT = Cantidad de tablas

FE = Factor de esfuerzo

PN = Prioridad del negocio (valor entre 1 y 10)

CC = Complejidad total del componente

Prioridad final del componente = ∑Factor de esfuerzo + ∑Prioridad Negocio

Tabla 29: Excel de priorización de los componentes

C	R	CI	CIT	E	CE	GN	CGN	CT	FE	PN	Prioridad final	CC
Nomencladores	Adicionar elemento de nomenclador	4	5	BaseNomAlias,BaseNomTipoetapa,BaseNomTipomodelo,BaseNomTipoplan	4	NomAliasController, NomTipoetapaController,NomTip	4	4	20	9	29	131

						omodeloController,NomTipoplanController						
	Modificar elemento de nomenclador	5	5	BaseNomAlias,BaseNomTipoetapa,BaseNomTipomodelo,BaseNomTipoplan	4	NomAliasController,NomTipoetapaController,NomTipomodeloController,NomTipoplanController	4	4	24	9	33	
	Eliminar elemento de nomenclador	5	5	BaseNomAlias,BaseNomTipoetapa,BaseNomTipomodelo,BaseNomTipoplan	4	NomAliasController,NomTipoetapaController,NomTipomodeloController,NomTipoplanController	4	4	20	6	26	
	Activar elemento de nomenclador	3	5	BaseNomAlias,BaseNomTipoetapa,BaseNomTipomodelo,BaseNo	4	NomAliasController,NomTipoetapaControl	4	4	10	4	14	

				mTipoplan		ler,NomTipomodeloController,NomTipoplanController						
	Desactivar elemento de nomenclador	3	5	BaseNomAlias,BaseNomTipoetapa,BaseNomTipomodelo,BaseNomTipoplan	4	NomAliasController, NomTipoetapaController,NomTipomodeloController,NomTipoplanController	4	4	10	4	14	
	Consultar elemento de nomenclador	3	5	BaseNomAlias,BaseNomTipoetapa,BaseNomTipomodelo,BaseNomTipoplan	4	NomAliasController, NomTipoetapaController,NomTipomodeloController,NomTipoplanController	4	4	10	5	15	
Indicadores	Adicionar indicador	10	7	BaseNomIndicador,BaseNomIndicadorexterno	2	GestindicadoresController	1	2	20	7	27	111

	Modificar indicador	9	7	BaseNomIndicador,BaseNomIndicadorexterno	2	GestindicadoresController	1	2	25	8	33	
	Eliminar indicador	8	7	BaseNomIndicador,BaseNomIndicadorexterno	2	GestindicadoresController	1	2	20	6	26	
	Consultar indicador	5	7	BaseNomIndicador,BaseNomIndicadorexterno	2	GestindicadoresController	1	2	20	5	25	
Plantilla	Adicionar plantilla	10	6	DatPlantilla	1	GesplantillaController	1	4	25	7	32	319
	Modificar plantilla	3	7	DatPlantilla	1	GesplantillaController	1	4	20	8	28	
	Eliminar plantilla	5	5	DatPlantilla	1	GesplantillaController	1	4	20	8	28	
	Activar plantilla	4	3	DatPlantilla	1	GesplantillaController	1	4	12	5	17	
	Desactivar plantilla	6	3	DatPlantilla	1	GesplantillaController	1	4	12	5	17	
	Consultar plantilla	1	4	DatPlantilla	1	GesplantillaController	1	4	10	5	15	
	Adicionar columna	2	6	NomColumna,NomTipocelda,DatCelda	3	GescolumnaController	1	1	20	7	27	
	Eliminar columna	4	6	NomColumna,NomTipocelda,DatC	3	GescolumnaControll	1	1	20	5	25	

				elda		er						
	Adicionar fila	6	7	DatPlantilla,NomTipocelda,DatCelda	3	GesplantillaController	1	4	20	8	28	
	Eliminar fila	7	7	DatPlantilla,NomTipocelda,DatCelda	3	GesplantillaController	1	4	15	8	23	
	Adicionar parámetros	7	4	DatPlantilla	1	GesplantillaController	1	4	20	7	27	
	Activar parámetros	7	4	DatPlantilla	1	GesplantillaController	1	4	10	8	18	
	Desactivar parámetros	7	3	DatPlantilla	1	GesplantillaController	1	4	10	7	17	
	Consultar parámetros	7	5	DatPlantilla	1	GesplantillaController	1	4	10	7	17	
Configuración de plan	Adicionar configuración	8	3	BaseDatDenominacionconf,BaseNomDenominacion,BaseDatConfconcreta,BaseDatConfetapa	4	ConstplanesController	1	4	15	10	25	128
	Modificar configuración	9	5	BaseDatDenominacionconf,BaseNomDenominacion,BaseDatConfconcreta,BaseDa	4	ConstplanesController	1	4	20	4	24	

				tConfetapa								
	Eliminar configuración	9	4	BaseDatDenominacionconf,BaseNomDenominacion,BaseDatConfconcreta,BaseDatConfetapa	4	ConstplanesController	1	4	15	5	20	
	Activar configuración	6	3	BaseDatDenominacionconf,BaseNomDenominacion,BaseDatConfconcreta,BaseDatConfetapa	4	ConstplanesController	1	4	20	5	25	
	Desactivar configuración	6	3	BaseDatDenominacionconf,BaseNomDenominacion,BaseDatConfconcreta,BaseDatConfetapa	4	ConstplanesController	1	4	20	3	23	
	Consultar configuración	7	5	BaseDatDenominacionconf,BaseNomDenominacion,BaseDatConfconcreta,BaseDatConfetapa	4	ConstplanesController	1	4	10	1	11	
Construcción de	Adicionar plan	2	9	BaseDatPlan	1	Constplan	1	2	20	8	28	108

plan						esControll er						
	Definir etapas asociadas	3	8	BaseDatEtapa	1	Constplan esControll er	1	2	15	8	23	
	Configurar celdas de modelo	5	7	BaseDatCeldam odelo,BaseDatC eldaoperacional, BaseDatModelo	3	Constplan esControll er	1	4	25	7	32	
	Adicionar modelo	1	4	BaseDatModelo	1	Constplan esControll er	1	1	20	5	25	
Realizacion de plan	Adicionar restricción	6	5	BaseDatRestricci on	1	Gestrestric cionContro ller	1	1	15	6	21	368
	Eliminar restricción	7	6	BaseDatRestricci on	1	Gestrestric cionContro ller	1	1	13	6		
	Consultar restricción	5	5	BaseDatRestricci on	1	Gestrestric cionContro ller,Gestm odeloContr oller	1	1	10	6	16	
	Validar restricción	5	3	BaseDatRestricci on	1	Gestrestric cionContro ller	1	1	15	5	20	

	Adicionar comentario	7	4	BaseDatComentariorechazado	1	GestmodeloController	1	1	10	4	14	
	Modificar comentario	7	5	BaseDatComentariorechazado	1	GestmodeloController	1	1	10	4	14	
	Consultar comentario	7	5	BaseDatComentariorechazado	1	GestmodeloController	1	1	10	9	19	
	Oficializar plan	7	8	BaseDatVersionplan	1	GestmodeloController	1	1	15	9	24	
	Modificar modelo	10	9	BaseDatVersionplanmodelo	1	GestmodeloController	1	1	20	10	30	
	Eliminar modelo	10	7	BaseDatVersionplanmodelo	1	GestmodeloController	1	1	20	9	29	
	Consultar modelo	9	5	BaseDatVersionplanmodelo	1	GestmodeloController	1	1	10	8	18	
	Terminar modelo	8	7	BaseDatVersionplanmodelo	1	GestmodeloController	1	1	10	10	20	
	Confirmar modelo	9	5	BaseDatVersionplanmodelo	1	GestmodeloController	1	1	10	6	16	
	Rechazar modelo	8	5	BaseDatVersionplanmodelo	1	GestmodeloController	1	1	10	6	16	
	Hacer oficial		8	BaseDatVersionplan	1	GestmodeloController	1	1	15	6	21	
	Copiar modelo	8	5	BaseDatVersionplanmodelo	1	GestmodeloController	1	1	18	8	26	

	Eliminar plan	9	5	BaseDatVersionplanmodelo,….	2	GestmodeloController	1	1	10	7	17	
	Consultar plan	8	5	BaseDatVersionplanmodelo,….	2	GestmodeloController	1	1	10	7	17	
	Iniciar planificación	10	1	BaseDatVersionplanmodelo,….	2	GestmodeloController	1	1	20	10	30	
Procesador matemático	Adicionar fórmula	8	5		0	EditorController	1	0	15	7	22	92
	Modificar fórmula	8	7		0	EditorController	1	0	20	9	29	
	Eliminar fórmula	7	5		0	EditorController	1	0	20	6	26	
	Consultar fórmula	6	4		0	EditorController	1	0	10	5	15	
	Validar fórmula	10	3		0	EditorController	1	0	23	8	31	
	Obtener valores de argumentos de la fórmula	10	6		0	EditorController	1	0	20	9	29	
	Obtener expresión polaca	10	8			EditorController	1	0	24	8	32	
	Evaluar función	10	5		0	EditorController	1	0	20	8	28	
Calculo de las		10	6	DatNecesidad	1	Calnecesid	1	1	25	9	34	292

Necesidades	CalcularNecesidades					adesController						
	Consolidar necesidades	10	8	DatNecesidad	1	CalnecesidadesController	1	1	20	9	29	
	Convertir UM	7	7	DatNecesidad	1	GestnormasController	1	1	20	9	29	
	Gestionar Actividad	7	5	NomActividad	1	GestionaractividadController	1	1	15	7	22	
	Gestionar asociación conceptos a criterios	8	5	DatConceptocriterio	2	AsociarcriconcController	1	1	15	8	23	
	Gestionar asociación criterios a actividades	8	7	DatCriterioactividad	2	AsociarcriactController	1	1	15	8	23	
	Gestionar Concepto	7	5	NomConcepto	1	GestionarconceptoController	1	1	15	7	22	
	Gestionar Criterio	7	5	NomCriterio	1	GestionarcriterioController	1	1	15	7	22	
	Gestionar Niveles de	8	9	DatNivelactividad	1	GesnivelactividadCon	1	1	20	7	27	

	Actividad					troller						
	Gestionar Normas	9	9	DatNorma	1	Gestnormas Controller	1	1	21	9	30	
	Gestionar valores capturadas	9	10	DatNorma	1	Gestnormas Controller	1	1	22	9	31	
Buscador	Realizar búsquedas	10	10		0	Consultar Controller	1	0	24	9	33	33

Anexo 6: Servicios de la integración interna de la línea.

Tabla 30: Servicios. Componente Procesador matemático

Procesador matemático	**Descripción**	**Usado por:**	**Usa:**	**Parámetros**	**rna**
1-DameResultado($formula)	Devuelve el resultado	Realización		Formula matemática	Arreglo de objetos

	de la fórmula	del plan			
2- DameFormula()	Devuelve la fórmula	Construcción del plan, Plantilla			Arreglo de objetos

Tabla 31: Servicios. Componente Configuración del plan

Configuración de plan	Descripción	Usado por:	Usa:	Parámetros	Retorna
1-plantillaUsada ($idplantilla)	Dar plantilla usada	Plantilla		Id de la plantilla	True o False plantilla usada o no
2-GetTodasConf()	Dar todas las configuraciones	Construcción de planes		no	Arreglo de objetos configuraciones
3-GetTodasEtapas($iddenominacion)	Dar las etapas dado id de una configuración	Construcción del plan	- De Nomencladores: getEtapasAsoc($Object1)	Id denominación	Arreglo de objetos todas las etapas
4-GetConfEtapas($idetapa)	Dar configuración de etapa dado id.	Construcción del plan		Id de etapa	Arreglo de objetos configuración de etapa
5-GetDenominacion($iddenominacion)	Dar denominación de una configuración	Construcción del plan		Id denominación	Arreglo de objetos denominación
6-GetIdtipoEtapa($arrIdconfetapa)	Dar tipo de etapa según id de configuración de etapa	Realización del plan		id de configuración de etapa	Arreglo de objetos tipo de etapa
7-planUsado($idtipoplan)	Plan usado o no, según id de tipo de plan			Id tipo de plan	True o False plan usada o no
8-etapaUsada($idtipoetapa)	Etapa usada según id de tipo de etapa			id de tipo de etapa	True o False etapa usada o no
9-getPlantillaByEtapa($idtipoetapa)	Dar plantilla por etapa dado id de tipo de etapa			id de tipo de etapa	Arreglo de objetos plantilla por etapa
10-getModelosByEtapaConf($idtipoetapa,$iddenominacion)	Dar Modelos por etapa dado id de etapa e id de denominación	Realización del plan	-De Plantilla: getIdModelos($Object) -De Nomecladores: getModelosFull($arrIdmodelos)	id de tipo de etapa, Id denominación	Arreglo de objetos modelos por etapa

11-getEtapaByIdconfEtapa($idconfetapa)	Dar etapa por id de configuración de etapa dado id de configuración de etapa	Construcción del plan			Arreglo de objeto

Tabla 32: Servicios. Componente Construcción del plan

Construcción de planes	Descripción	Usado por:	Usa:	Parámetros	Retorna
1-Dameplanes()	Devolver todos los planes	Realización del plan, Buscador			Arreglo de objetos de planes.
2-DameplanesGrid($limite ,$inicio)	Para cargar los planes en un grid.			Inicio y fin	Arreglo de objetos de planes.
3-CantDameplanes()	Devolver la cantidad de planes	Buscador			Arreglo de objetos la cantidad de planes.
4-DameEtapas($idplan)	Devolver todas las etapas	Realización del plan		Id del plan	Arreglo de objetos de etapas.
5-cambiarEstadoModelo($idmodelo, $idestadomodelo)	Para cambiar el estado a los modelos	Realización del plan		Id del modelo e idestadomodelo.	True o False si se cambio el estado o no.
6-cerrarEtapa($idetapa)	Para cerrar las etapas	Realización del plan		Id de la etapa	True o False si se cambio la etapa o no.
7-DameModelos($idversion, $idplan, $idetapa)	Para obtener todos los modelos dado los parámetros que se pasan			Id de la version , id del plan e id de la etapa	Arreglo de objetos de modelos.
8-DameModelosporEtapa($idetapa, $idversion)	Devolver todos los modelos dada una etapa determinada.	Realización del plan		Id de la etapa e id de la version	Arreglo de objetos de modelos.
9-DameEjerPlanes($id)	Devolver los ejercicios por los que pasará el plan.	Buscador		Id del plan	Arreglo de objetos de ejercicios.
10-DameEjercicios()	Devolver todos los ejercicios	Buscador			Arreglo de objetos de ejercicios.
11-DameVersionPorId($idmodelo)	Devolver las versiones dado el id del modelo			Id del modelo	Arreglo de objetos de versiones.
12-DamePlantillaPorId($idmodelo)	Devolver la plantilla dado el id del modelo.	Construcción del plan, Realización del plan, Procesador matemático		Id del modelo	Arreglo de objetos de plantillas.
13-DameModelosporPlan($idplan)	Devolver los modelos que tiene un	Procesador		Id del plan	Arreglo de objetos

ANEXOS

	plan.	matemático			de modelos.
14-DamePlanesEjer($idejercicio)	Devolver los planes dado un ejercicio.	Procesador matemático, Buscador		Id del ejercicio	Arreglo de objetos de planes.
15-DatosModeloporIdModelo($idmodelo)	Devolver los datos de los modelos dado el id de un modelo.			Id del modelo	Arreglo de objetos con los datos del modelo.
16-DatosModelo($idmodelo)	Devolver los datos de los modelos			Id del modelo	Arreglo de objetos con los datos del modelo.
17-sayIfUsedConf($iddenominacion)	Devolver si una configuración determinada.	Configuración del plan		Id de la denominacion	Arreglo de objetos.
18-DatosModeloPlan()	Devolver los datos de los modelos de un plan.	Buscador			Arreglo de objetos con los datos del modelo.
19-DatosModeloPlanGrid($limit ,$start)	Devolver los datos de los modelos de un plan.			Inicio y fin	Arreglo de objetos con los datos del modelo.
20-dameValorCeldas($idmodelo)	Devolver el valor de las celdas dado el id de un modelo.	Realización del plan	De Plantilla : DamePosCelda($idcelda) y getIdTipoCelda($idcelda)	Id del modelo	Arreglo de objetos con los datos de las celdas.
21-dameValorCelda($idcelda)	Devolver el valor de las celdas de un modelo.	Realización del plan		Id de la celda	Arreglo de objetos con los datos de las celdas.
22-Duplicar($idplan,$idmodelo,$idversion)	Duplicar el modelo de un plan	Realización del plan		Id del modelo, id de la version	True o False si se duplicó el modelo.
23-EliminarModelo($idmodelo)	Eliminar un modelo del plan	Realización del plan		Id del modelo	True o False si se eliminó el modelo.
24-Buscar($idplan)	Buscar un plan determinado	Buscador		Id del plan	True o False si encontró el plan.
25-Buscar1($arrayIdplan)	Buscar un plan determinado dentro de un arreglo.			Id de la versión	True o False si encontró el plan.
26-PlanporIdversion($idversion)	Devolver los planes dada una versión.			Id de la versión	Arreglo de objetos con los planes.

ANEXOS

27-PlanporCodigo($codigo)	Devolver los planes dado un código.			código	Arreglo de objetos con los planes.
28-PlanporIdTipoplan($idtipoplan)	Devolver el plan.			Id del tipo de plan	Arreglo de objetos con los planes.
29-PlanporArrayIdPlan($arrayidplan)	Devolver el plan.			Objeto de planes	Arreglo de objetos con los planes.
30-buscarporEjyPlan($idplan,$idejercicio)	Devolver el plan			Id del plan , id del ejercicio	True o False
31-getPlanByIdyCodigo($codigo, $idplan)	Devolver el plan			Código , id del plan	Arreglo de objetos con los planes.
32-DameEstadosModelos()	Devolver el estado de los modelos.	Buscador			Arreglo de objetos con los estados de los modelos.
33-ModelosAcept($modelos =array())	Devolver los modelos en estado Aceptado.	Realización del plan		Objeto de modelos	Arreglo de objetos con los modelos que han sido aceptados.
34-ModifModelos($arrayCell)	Para modificar los modelos.	Realización del plan		Objeto de celdas	True o False si se modificó el modelo.
35-DameIdplantilla($idplan)	Devolver el id de una plantilla dado el id de un plan.	buscador		Id del plan	Arreglo de objetos con los ids de las plantillas.
36-DatosModeloPlanPorID($limit ,$start, $idplan)	Devolver los datos de los modelos.			Inicio y fin , id del plan	Arreglo de objetos con los datos del modelo.
37-datosIDPlantilla($limit ,$start, $arridplantilla)	Devolver los datos de la plantilla.			Inicio y fin , objeto de plantilla	Arreglo de objetos con los datos de la plantilla.
38-datosModeloByIdPlantilla($arridplantilla)	Devolver los datos de los modelos.	Buscador		objeto de plantilla	Arreglo de objetos con los datos del modelo.
39-DameEstModByIdEstMod($arridestadomodelo)	Devolver el estado de los modelos.	Buscador		objeto de estado de los modelos	Arreglo de objeto
40-DatosIdversionplan($limit ,$start, $idmodelo)	Devolver versiones del plan.			Inicio y fin , id del modelo	Arreglo de objeto
41-DatosIdestadomodelo($limit ,$start, $idestadomodelo)	Devolver estado de los modelos.			Inicio y fin , id estadomodelo	Arreglo de objeto
42-DatosIdversion($limit ,$start, $idversion)	Devolver datos.			Inicio y fin , id de la version	Arreglo de objeto

ANEXOS

43-Datos($limit ,$start, $idplan,$idversion,$idestadomodelo,$idmodelo,$arridplantilla)	Devolver datos.			Inicio y fin , id del plan, id estadomodelo, id version , id del modelo , objeto de plantilla	Arreglo de objeto
44-DamedatosSinIdversion($limit ,$start, $idplan,$idestadomodelo,$idmodelo,$arridplantilla)	Devolver datos.			Inicio y fin , id del plan, id estadomodelo, id version , id del modelo , objeto de plantilla	Arreglo de objeto
45-DamedatosSinVersiones($limit ,$start, $idplan,$idestadomodelo,$arridplantilla)	Devolver datos.			Inicio y fin , id del plan, id estadomodelo, id version , ,objeto de plantilla	Arreglo de objeto
46-DamePlanModelo($limit ,$start, $idplan,$idmodelo, $arridplantilla)	Devolver plan.			Inicio y fin , id del plan, id estadomodelo, id version , ,objeto de plantilla	Arreglo de objeto
47-DameIdversionplan($idmodelo)	Devolver id de la versión de un plan.			Id del modelo	Arreglo de objeto
48-getPlanByIdyTPlan($idtipoplan, $idplan)	Devolver plan.			Id del tipo de plan , id del plan	Arreglo de objeto
49-DameDatosSinIdVerPlanMod($limit ,$start, $idplan,$idversion,$idestadomodelo,$arridplantilla)	Devolver datos del modelo.			Inicio y fin , id del plan, id estadomodelo, id version , ,objeto de plantilla	Arreglo de objeto
50-DameVersionPlan($idplan)	Devolver las versiones de un plan determinado	Realización del plan		Id del plan	Arreglo de objeto con las versiones de un plan
51-getVersionbyIdVersion($idversion)	Devolver una versión determinada dado el id de una versión	Buscador		Id version	Arreglo de objeto
52-getVersionPlanModelo($idmodelo)	Obtener las versiones de un plan.	Buscador		Id modelo	Arreglo de objeto
53-DameTodosIdModelos($versionplan)	Devolver los id de los modelos de una versión determinada.	Buscador		Id del modelo	Arreglo de objeto
54-DameTodosIdModelo()	Devolver todos los id de los modelos.			Version del plan	Arreglo de objeto

55-DameMod($ver)	Devolver modelos.				Arreglo de objeto
56-DameIdmodeloversion($idversionplan)	Devolver id de los modelos.			Id de la Version del plan	Arreglo de objeto
57-DameDatosPlanModelo($limit ,$start, $idplan,$arridplantilla)	Devolver los datos de un plan.			Inicio y fin , id del plan y objeto de plantilla	Arreglo de objeto
58-getVersionbyIdPlan($idplan)	Devolver versión.	Buscador		Id del plan	Arreglo de objeto

Tabla 33: Servicios. Componente Realización del plan

Realización del Plan	Descripción	Usado por:	Usa:	Parámetros	Retorna
1-getUltimaVersionbyIdPlan($idplan)	Devolver la ultima versión del plan.			Id del plan	Arreglo de objetos
2- IndicadorenUso($idindicador)	Devolver que indicador se esta usando.	Indicadores		Id Indicador	Arreglo de objetos
3-getIdPlanByIdVersion($idversion)	Devolver el id del plan.			Id version	Arreglo de objetos
4-DameEstadoPlan($idversionplan)	Devolver el estado de un plan.	Buscador		Id versionplan	Arreglo de objetos
5- dameIdPlanIfAprobado($aprobado)	Devolver id del plan si esta en estado aprobado.			aprobado	Arreglo de objetos
6- getIdPlanByArrayIdVersion($idversionplan)	Devolver plan.			Id versionplan	Arreglo de objetos
7- dameIdPlanIfOficial($oficial)	Devolver id si el plan es oficial.			oficial	Arreglo de objetos
8- getIdPlanBIdVersion($idversionplan, $idplan)	Devolver id del plan.	Buscador		Id versionplan , Id del plan	Arreglo de objetos
9- GetEstadoByIdVPlanYaprobado($idversionplan, $estado)	Devolver planes que estén aprobados.			Id versionplan , estado	Arreglo de objetos
10- GetEstadoByIdVPlanYoficial($idversionplan, $oficial)	Obtener estado del plan.			Id versionplan , oficial	Arreglo de objetos
11- getVersionAll($idversionplan)	Obtener todas las versiones.			Id versionplan	Arreglo de objetos

ANEXOS

Tabla 34: Servicios. Componente Nomencladores

Nomencladores	Descripción	Usado por:	Usa:	Parámetros	Retorna
1- obtenerTipoModeloDenomporPlantilla($idtipo)	Obtener la denominación de los tipos de modelos.	Buscador		Id tipo de modelo	Arreglo de objetos
2- obtenerTipoModeloDenom($idtipo)	Obtener la denominación de los tipos de modelos.	Construcción del plan, Buscador		Id Tipo de modelo	Arreglo de objetos
3- obtenerTipoEtapaDenom($idtipo)	Obtener la denominación de los tipos de etapas.	Construcción del plan		Id tipo	Arreglo de objetos
4- obtenerTodosmodelo	Obtener todos los modelos.	Configuración del plan, Plantilla			Arreglo de objetos
5- obtenerTodosEtapa	Obtener todas las etapas.	Configuración del plan			Arreglo de objetos
6- obtenerTodosplan	Obtener todos los planes.	Configuración del plan, construcción del plan			Arreglo de objetos
7- obtenerTipoPlanDenom	Obtener tipo de plan.	Buscador			Arreglo de objetos
8- obtenerPlanporID($idtipoplan)	Obtener plan.	Configuración del plan, construcción del plan		Id tipo plan	Arreglo de objetos
9- getEtapasDesAsoc($ar)	Obtener las etapas que no están asociadas a algún plan.	Configuración del plan		Objeto	Arreglo de objetos
10-getEtapasAsoc($arr)	Obtener las etapas que están asociadas a algún plan.	Configuración del plan, Realización del plan		Objeto	Arreglo de objetos
11- getModelosFull($ar)	Obtener todos los modelos.	Configuración del plan		Objeto	Arreglo de objetos

12- getModelosNoAsocFull($ar)	Obtener modelos no asociados a ningún plan.	Configuración del plan		Objeto	Arreglo de objetos
13- getPlanByArr($ar)	Obtener plan.	Configuración del plan, Buscador		Objeto	Arreglo de objetos
14- getPlanByArr1($idplan)	Obtener plan.	Configuración del plan		Id del plan	Arreglo de objetos

Tabla 35: Servicios. Componente Indicadores

Indicadores	**Descripción**	**Usado por:**	**Usa:**	**Parámetros**	**Retorna**
1- getIndicadoresActivados($padre)	Obtener indicadores activados.	Plantilla, Procesador matemático		Padre, numero entero	Arreglo de objetos
2- getIndicadoresNormados	Obtener indicadores normados.				Arreglo de objetos
3- getIndicadorById($idindicador)	Obtener indicador.	Realización del plan, Procesador matemático		Id del indicador	Arreglo de objetos
4- ObtenerAtributoEstado($limit,$start, $idindicador)	Obtener atributos.			Comienzo , fin e id del indicador	Arreglo de objetos
5- isLeaf($idindicador)	Devolver si un indicador es hoja.			Id del indicador	boolean
6- isValid($codigo)	Devolver si un código es válido o no.			codigo	boolean
7- obtenerObjetos	Devolver objetos.				Arreglo de objetos
8- obtenerListaElement($idobjeto)	Obtener lista de elementos.			Id objeto	Arreglo de objetos
9- getActivados	Devolver indicadores activados.	Construcción del plan, Plantilla			Arreglo de objetos
10- getIndicadoresArray($limit,$start)	Devolver indicadores.			Comienzo , fin	Arreglo de objetos
11- getPadresHijos($idindicador)	Devolver hijos de un indicador.	Procesador matemático		Id del indicador	Arreglo de objetos

12- getPorCodigo($codigo)	Devolver indicador dado un código.			codigo	Arreglo de objetos
13- getValInd($idindicador, $atributo)	Devolver valor del indicador.			Id indicador y atributo	Arreglo de objetos

Tabla 36: Servicios. Componente Plantilla

Plantilla	Descripción	Usado por:	Usa:	Parámetros	Retorna
1- isUsed($idindicador)	Devolver si se está usando un indicador determinado.	Indicadores		Id del indicador	boolean
2- getIdModelos($arr)	Devolver id de los modelos.	Configuración del plan		Objeto	Arreglo de objetos
3- getPlantillasByIdModelos($arrModelos)	Devolver plantillas.	Construcción del plan, Buscador		Objeto	Arreglo de objetos
4- getPlantillasbyidmodelo($idmodelo)	Devolver plantillas.	Configuración del plan		Id del modelo	Arreglo de objetos
5- getColumnas	Devolver las columnas.	Realización del plan			Arreglo de objetos
6- tipoModeloUsado(idtipomodelo)	Devolver el tipo de modelo que se esta usando.			idtipomodelo	Arreglo de objetos
7- dameReader(idplantilla)	Devolver el reader para la confección del modelo.	Construcción del plan, Realización del plan		Id de la plantilla	Arreglo de objetos
8- dameColumnasP($idplantilla)	Devolver las columnas de una plantilla determinada.	Construcción del plan, Realización del plan, Procesador matemático		Id de la plantilla	Arreglo de objetos
9- dameConfEdicCelda($idplantilla)	Devolver la configuración de las celdas que son editadas.	Realización del plan		Id de la plantilla	Arreglo de objetos
10- dameIndicadores($idplantilla)	Devolver indicadores.	Construcción del plan, Realización del plan, Procesador		Id de la plantilla	Arreglo de objetos

ANEXOS

		matemático			
11- getPlantilla	Devolver plantilla.	Configuración del plan		Id de la plantilla	Arreglo de objetos
12- GetIndCol(idplantilla)	Devolver indicador de una columna.			Id de la plantilla	Arreglo de objetos
13- GetIdtipoModelo($arrIdplantilla)	Devolver id de un tipo de modelo.	Realización del plan		Objeto	Arreglo de objetos
14- ConfEdicCelda($idplantilla)	Devolver las celdas que son editadas.			Id de la plantilla	Arreglo de objetos
15- DameTipoModelos	Devolver tipos de modelos.	Buscador			Arreglo de objetos
16- DameColumnasInd($idindicador)	Devolver columnas.	Procesador matemático		Id del indicador	
17- getIdTipoCelda($idcelda)	Devolver id del tipo de celda.			Id de la celda	Arreglo de objetos
18- getTipoModelobyidplantilla($idplantilla)	Devolver tipo de modelo.	Construcción del plan		Id de la plantilla	Arreglo de objetos
19-DamePosCelda($idcelda)	Devolver la posición de una celda.			Id de la celda	Arreglo de objetos
20-DameIdCelda($idplantilla, $idindicador, $idcolumna)	Devolver id de la celda.	Procesador matemático		Id de la plantilla, id del Indicador , Id de la columnas	Arreglo de objetos
21- DamePlantillaCod($codigo)	Devolver plantilla.			Código de la plantilla	Arreglo de objetos
21- DamePlantillaCod($codigo, $idtipomodelo)	Devolver plantilla.			Código e Id del tipo de modelo	Arreglo de objetos
22- getColumna($idcolumna)	Devolver la columna.			Id de la columna	Arreglo de objetos
23- ObtenerIdTipoModelo($idplantilla)	Devolver el id del tipo de modelo.	Buscador		Id de la plantilla	Arreglo de objetos
24- ObtenerIdPlantilla($idtipomodelo)	Devolver el id de la plantilla.	Buscador		Id del tipo de modelo	Arreglo de objetos
25- DameIdplantilla($idtipomodelo)	Devolver el id de la plantilla.			Id del tipo de modelo	Arreglo de objetos