Universidad de las Ciencias Informáticas

"Facultad 3"

Título: "Análisis y diseño del componente alertas y avisos del Marco de Trabajo Sauxe."

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

Autor: Mayté Sánchez Fernández.

Tutores: Ing. Pedro Manuel Nogales Cobas.

Ing. Javier Ruiz Duran.

Junio 2011

Un ingeniero no es una copia, es original y se atreve a cambiar una realidad, no importa el tiempo o el espacio, todo es posible mientras crea que es así.

DEDICATORIA

Dedicatoria Especial.

Mi madre, por haberme dado la vida, por ser siempre mi primera opción cuando creo que todo está perdido por lo buenos y malos ratos que hemos vivido juntas por ser esa persona que siempre ha confiado en mí y la que me ha impulsado, dado aliento y apoyo incondicional y sobre todo por ser siempre mi razón para seguir adelante.

Dedico esta tesis a:

Mi padre, por haberme enseñado que cada día es un nuevo comienzo y por haberme regalado una niñez llena de amor y cariño.

A mi hermanita porque es lo más grande que tengo en este mundo.

Mi abuela Hilda, por quererme tanto.

Mi novio, por todo el amor que día a día me das, por mimarme tanto, por aguantar mis malcriadeces, por obligarme a despertar por las mañanas ,por ser mi alegría, por ayudarme y ser incondicional en todo lo que me ha hecho falta en este maravilloso tiempo que hemos estado juntos.

RESUMEN

En el mundo existen numerosos Marcos de Trabajo que brindan un conjunto de componentes genéricos reutilizables que agilizan el proceso de desarrollo de software. Un componente muy importante con el que deben contar los Marcos de Trabajo por las ventajas que brinda es el de Alertas y Avisos. Actualmente el Centro de Informatización de la Gestión de Entidades está desarrollando un Marco de Trabajo para aplicaciones web de gestión denominado Sauxe, el cual debe contar con un sistema de Alertas y Avisos, ya que estos brindan facilidades enfocadas al conocimiento de los eventos que ocurren en el sistema en tiempo real, agilizando de esta manera tareas que tienden a ser complejas, evitando cuantiosas pérdidas de tiempo, y permitiendo en todo momento un control del estado de las actividades de negocio y trabajo. Su misión es brindar comunicación en tiempo real entre la PC cliente y el servidor logrando así una gran mejoría de los procesos dentro de los restantes componentes que integran el Marco de Trabajo Sauxe.

La intención fundamental de este trabajo es lograr una solución para el tratamiento de las particularidades que requiere un sistema de Alertas y Avisos que se adecue al Marco de Trabajo que se desarrolla. Para esto se pretende elaborar el análisis y diseño que incluye entender el dominio de la información del problema, definir las funciones que debe realizar el componente y dividir en forma jerárquica los modelos que representan la información, funciones y comportamiento. Su impacto se concentra en el establecimiento de las bases para la posterior implementación del mismo, teniendo en cuenta los resultados que aquí se reseñan, siendo un recurso que se ha concebido para satisfacer las necesidades del cliente en el mayor grado posible.

Palabras claves: Alertas y Avisos, Componentes, Marco de Trabajo

CONTENIDO

Figuras

Tablas

INTRODUCCIÓN

El avance de las Tecnologías de la Información y las Comunicaciones (TIC), ha acarreado la creación de diversas herramientas con el fin de agilizar el proceso de desarrollo de aplicaciones web de gestión, entre ellas se encuentran los Marcos de Trabajo; para el desarrollo rápido de aplicaciones. Estos cuentan con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado. Incluyen además un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Dados los componentes incluidos que posee libera al programador de la escritura de código de bajo nivel. Y permiten la portabilidad de aplicaciones de una arquitectura a otra. (1)

Cuba no está ajena a estos avances y en la actualidad cuenta con instituciones que se dedican a implementar Marcos de Trabajo con el fin de facilitar la implementación de aplicaciones web de gestión.

Una de estas instituciones es el Centro de Informatización de la Gestión de Entidades (CEIGE) de la Universidad de las Ciencias Informáticas (UCI) que hoy en día es puntero del desarrollo tecnológico en cuba; este centro tiene como una de sus tareas hacer un Marco de Trabajo. Para ello se creó un grupo central de arquitectura que fue definiendo por cada capa, los componentes que formarían parte de la línea base. De estos componentes, algunos fueron reutilizados y otros extendidos de Zend Framework para dar lugar al Marco de Trabajo que adoptó el nombre de Sauxe. Este contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. A medida que el sistema Sauxe avanza, mayores son los requerimientos que debe cumplir la arquitectura para soportar y brindar las prestaciones requeridas. Actualmente por la propia naturaleza de la arquitectura cliente-servidor empleada por las aplicaciones web no se puede notificar a los clientes de algún evento existente en el servidor en tiempo real.

Dada esta situación, se define como **Problema a resolver**: El equipo de desarrollo del componente Alertas y avisos, no cuenta con los elementos necesarios para su posterior implementación. Para darle solución al problema planteado se define como **objeto de estudio** Marcos de Trabajo para el desarrollo de aplicaciones web de gestión el **campo de acción** de la investigación se enfoca específicamente en el

Marco de Trabajo Sauxe, teniendo como **objetivo general:** Realizar el análisis y diseño del componente de Alertas y Avisos del Marco de Trabajo Sauxe. Del cual se deriva como **Objetivos Específicos**:

1. Construir el marco teórico de la investigación.
2. Realizar el análisis y diseño de la solución.
3. Validar la solución propuesta.

La **idea a defender** de la investigación es que la realización del análisis y diseño del componente Alertas y Avisos, permitirá que se pueda implementar dicho componente en función de lograr notificaciones en tiempo real en los sistemas que utilizan el Marco de Trabajo Sauxe. Las **tareas** a realizar para el trabajo de investigación son:

1. Estudiar el estado del arte de los Marcos de Trabajo existentes.
2. Estudiar validación de requisitos.
3. Estudiar las técnicas y métricas para validar el diseño.
4. Definir las reglas del negocio.
5. Realizar mapa de procesos.
6. Realizar la descripción del negocio.
7. Realizar el modelo conceptual.
8. Realizar la especificación de los requisitos funcionales.
9. Validar los requisitos funcionales.
10. Realizar el prototipo de IU.
11. Aplicar patrones de diseño.
12. Realizar diagramas de clases del diseño.
13. Realizar descripción de las clases del diseño.
14. Realizar el diagrama de componentes.
15. Realizar la validación del diseño.

Resultados esperados: Con el análisis y diseño del componente Alertas y Avisos se podrán contar con una serie de artefactos que servirán como punto de partida para una posterior implementación del mismo, estos artefactos son los que propone específicamente el modelo orientado a componente del proyecto ERP-Cuba como son el modelo de procesos de negocio, la descripción de procesos de negocio, el modelo

conceptual, los prototipo de IU, la especificación de requisitos, los diagramas de clases, el diagrama de datos y finalmente los diagramas de componentes.

Los siguientes **métodos teóricos** sustentan la investigación:

Histórico-Lógico: Su empleo permitió el desarrollo evolutivo y coherente en el estudio de la ingeniería de requisitos y diseño, patrones de diseño, herramientas Case y Marcos de Trabajo para el desarrollo de los artefactos que proponen los flujos estudiados.

Analítico-Sintético: Permitió integrar y descomponer el conocimiento, descubriendo las relaciones para la utilización de artefactos propuestos por constituir este método una unidad dialéctica, determinando los aspectos esenciales y el arribo a conclusiones prácticas y teóricas.

Inductivo-deductivo: Permitió pasar de lo particular a lo general y viceversa favoreciendo objetivamente el enlace que se establece en la realidad entre lo singular y lo general ya que ambas se complementan mutuamente en el proceso de desarrollo científico.

Modelación: Pues se crean abstracciones que explican la realidad, por ejemplo, todos los modelos y diagramas presentados.

El documento se divide en 3 capítulos

Capítulo 1: Este capítulo presenta un estudio del estado del arte sobre los Marcos de Trabajo del mundo definiendo específicamente Sauxe. Se realiza una caracterización del modelo de desarrollo empleado, así como de la herramienta utilizada para desarrollar los artefactos. Se presenta además una breve descripción las tecnologías empleadas.

Capítulo 2: En este capítulo se realiza una caracterización del negocio propuesto. Se exponen los artefactos generados producto del análisis de los principales procesos del componente Alertas y Avisos y su descripción. Se emplea y explica la estrategia de captura de requisitos y modelado del sistema,

representando los artefactos que se generan para ello según la metodología usada. Además se realiza el modelado del diseño, donde se exponen los artefactos que lo componen, como subsistemas, diagramas de clases y arquitectura.

Capítulo 3. Se aplican, los patrones de diseño y las métricas que se utilizan para evaluar la calidad de los requerimientos capturados y el diseño realizado.

Este documento posee, además, Conclusiones por cada capítulo, Conclusiones generales, Recomendaciones, Referencias bibliográficas y Bibliografía.

Capítulo 1: Fundamentación teórica

INTRODUCCIÓN

En el capítulo se aborda un estudio del estado del arte de los Marcos de Trabajo que utilizan sistemas de notificación, las herramientas, lenguaje de modelado y modelo de desarrollo de software a utilizar durante la realización práctica del trabajo. Además se fundamentan la utilización de las mismas.

1. Aplicaciones Web de gestión

Es una página web especial, que tiene una base de datos asociada y que permite una mayor interacción del usuario; permiten agilizar las tareas administrativas de una empresa pudiendo reproducir e informatizar los procesos de trabajo del negocio. Estas aportan un considerable ahorro de tiempo, al evitar labores repetitivas, errores en trabajos rutinarios, etc. Además el modelado informático contribuye a la racionalización de sus procesos y permite en todo momento un control del estado de las actividades.

Otra ventaja es que la aplicación puede estar instalada en un servidor de internet, de esa manera será posible acceder y trabajar en ella desde otras sucursales, desde el domicilio particular o con un ordenador portátil desde cualquier ubicación (hoteles, ferias, congresos) en que haya conexión. (2)

Existen disímiles aplicaciones web de gestión entre ellas se encuentran los ERP (Planificación de Recursos Empresariales) el Redmine [1], ProjectPier [2], Collabtive [3], eGroupware[4], entre otras que contribuyen de una manera u otra a informatizar procesos manuales complejos logrando así la mejoría de los mismos.

Con el objetivo de ahorrar tiempo a los desarrolladores y desarrollar aplicaciones web que sean fáciles de mantener y rindan bajo mucha carga surgen los Marcos de Trabajo.

[1] **Redmine** es una herramienta para la gestión de proyectos y el seguimiento de errores.
[2] **ProjectPier** es una herramienta para gestionar proyectos, tareas y equipos de trabajo en las empresas mediante una herramienta web
[3] **Collabtive** es una herramienta para la administración de proyectos y grupos de trabajo a través de la web.
[4] **eGroupware** es una solución de trabajo en grupo vía web.

1.1. Marcos de Trabajo.

Los Marcos de Trabajo son desarrollados con el objetivo de brindarles a los programadores y diseñadores una mejor organización y estructura a sus proyectos. Brindan facilidades a la hora de optimizar código, y estandarizar la programación de una manera eficiente. Como principales y más usados se destacan: Symfony, CodeIgniter, CakePHP, Ruby on Rails, Django y Zend Framework. (3)

1.2 Marcos de Trabajo que utilizan sistemas de notificación.

1.2.1 Django

Django es un marco de desarrollo web sobre Python que permite desarrollar rápidamente aplicaciones web. Este presenta uno de los sistemas de plantillas más flexibles y potente que existen, separando la lógica de la presentación, desalentando la redundancia, permitiendo la seguridad y extensibilidad así como el fácil desacoplo del código HTML, Una de las principales ventajas de Django, es que no se interpone, dejando a los desarrolladores trabajar fuera del ámbito del framework según lo necesiten. A continuación se listan otras de sus ventajas:

Potente ORM (Mapeador de Objeto Relacional), infinita flexibilidad, eficiencia de SQL (lenguaje de consulta estructurado), sintaxis concisa y poderosa, opción de escribir SQL crudo cuando se necesita, bajo acoplamiento en el diseño de URLs (Localizador Uniforme de Recursos), uso de objetos request y permite el desarrollo en equipo. (4)

Desventajas de Django:

- Esta implementado en Python.
- No es tan simple de implantar.
- Es más lento que un framework en un lenguaje compilado.
- No incluye AJAX de serie. Su curva de aprendizaje suele ser un poco más elevada, aunque esto es en gran medida por su flexibilidad. Esa curva de aprendizaje elevada está dada principalmente por su configuración y la alta flexibilidad en el diseño de URLs.
- El sistema de plantilla de Django es ideal para los diseñadores no para programadores, por lo que no permite de manera intencional la inclusión directa de código de programación en HTML (Lenguaje de Marcado de Hipertexto), así como la creación de nuestras propias extensiones. (4)

Este Marco de Trabajo cuenta además con un sistema de notificación que permite al resto de las aplicaciones de un proyecto enviar notas a los usuarios. Esta nota puede ser recibida por correo o mostrarse en el sistema cuando el usuario inicia su sesión. El sistema de mensajes es una forma muy ligera y sencilla de enviarle mensajes a un usuario. Cada usuario tiene asociada una cola de mensajes, de forma que los mensajes lleguen en el orden en que fueron enviados. Los mensajes no tienen ni fecha de caducidad ni fecha de envío. La interfaz de administración de Django usa los mensajes para notificar que determinadas acciones han podido ser llevadas a cabo con éxito. Por ejemplo, al crear un objeto, se verá que aparece un mensaje en lo alto de la página de administración, indicando que se ha podido crear el objeto sin problemas. (4) Para esto utiliza la librería Django-celery. Esta es una librería que permite crear tareas de trabajo asíncronas gestionadas por un gestor de colas que está basada en el envío de mensajes de manera distribuida. Se focaliza en operaciones en tiempo real pero también soporta la calendarización de tareas, es decir, puede lanzar tareas que se tengan que ejecutar en un momento determinado o de manera periódica. Las unidades de ejecución, llamadas tareas, se ejecutan de manera concurrente en uno o más nodos de trabajo. Estas tareas pueden ejecutarse de manera asíncrona o bien de manera síncrona (esperando hasta que la tarea esté lista).

1.2.2 Web2py

web2py es un Marco de Trabajo de código abierto para el desarrollo ágil de aplicaciones basadas en web, está escrito en Python. web2py es un completo framework, lo que significa que contiene todos los componentes necesarios para construir plenamente aplicaciones web funcionales. Está diseñado para guiar a un desarrollador web a aplicar buenas prácticas de ingeniería de software, tales como el uso del Modelo Vista Controlador (MVC). Proporciona bibliotecas para ayudar a los desarrolladores a diseñar, implementar y probar cada una de estas tres partes por separado, y los hace trabajar juntos. Se construye para la seguridad. Esto significa que automáticamente resuelve muchos de los problemas que pueden dar lugar a vulnerabilidades de seguridad, siguiendo las prácticas establecidas, así, por ejemplo, valida todas las entradas (para evitar las inyecciones). Cambia el nombre de los archivos cargados (para prevenir ataques de directorio transversal). Incluye una base de datos Capa de abstracción (DAL) que escribe SQL dinámicamente. web2py es de código abierto y liberado bajo la licencia GPL2.0, pero las aplicaciones desarrolladas con este no están sujetas a ninguna restricción de licencia. Siempre y cuando no contengan el código fuente web2py, no se consideran "Obra derivada".

Desventajas

- Su principal desventaja radica en su falta de flexibilidad, porque a diferencia de Django las aplicaciones de web2py no son independientes del framework, ya que sus aplicaciones radican dentro del mismo, esto podría provocar problemas a la hora de acoplarlo con otro sistemas.

Este framework permite enviar SMS de notificación a los usuarios o clientes del mismo. Enviar mensajes SMS desde una aplicación web2py requiere un servicio de terceros que puede reenviar los mensajes al receptor. Por lo general esto no es un servicio gratuito, difiere de un país a otro. Este utiliza las empresas de telefonía para él envió del SMS. Cada compañía telefónica tiene una dirección de correo electrónico asociado únicamente a cada número de teléfono celular, los SMS se pueden enviar como un correo electrónico a ese número de teléfono. web2py viene con un módulo para ayudar en este proceso:
SMSCODES es un diccionario que mapea los nombres de las compañías telefónicas importantes. La función sms_email tiene un número de teléfono (en forma de cadena) y el nombre de una compañía telefónica y devuelve la dirección de correo electrónico del teléfono. (5)

1.2.3 Ruby on Rails

Ruby es un lenguaje de programación orientado a objetos puro. Rails es un framework de código abierto para Ruby que sirve para desarrollar aplicaciones web que acceden a bases de datos.

Rails es un framework de aplicaciones web que funciona en tiempo de ejecución pero además es una serie de scripts de ayuda que computarizan muchas de las tareas que hay que hacer cuando se desarrolla una aplicación web. Rails trata por todos los medios de minimizar el número de decisiones que tienes que tomar así como de eliminar el trabajo innecesario.

Desventajas

- La forma de estructurar el código, será la que el propio Framework dicte, ya que separa en carpetas cada capa de su modelo de desarrollo.

Características de Ruby on Rails

- Totalmente orientado a objetos
- Compatible con XMPP (Protocolo extensible de mensajería y comunicación de presencia)
- Rosca, basado en eventos
- Utilización de software bien conocido y bien probado como REXML (procesador de XML para el lenguaje de programación Ruby), en lugar de reinventar la rueda
- Muy fácil de ampliar
- Licencia de Ruby, que es compatible con la GNU GPL a través de una cláusula explícita de doble concesión de licencias.

Ruby utiliza el XMPP que no es más que "un protocolo abierto, XML de inspiración en tiempo real, mensajería instantánea extensible e información de presencia." Es utilizado por Jabber, el proyecto Gizmo, Google Talk, Pidgin, Kopete, y todo tipo de fuente abierta aplicaciones de mensajería instantánea. También puede ser utilizado por cualquier aplicación que quiera desarrollar un sistema de notificación para pasar mensajes de ida y vuelta. Este utiliza principalmente la biblioteca XMPP4R que es un XMPP / Jabber para la colección de Ruby, básicamente se puede hacer todo lo que desees, XMPP4R. Es totalmente compatible con XMPP, y también una amplia gama de extensiones XEPs (Extensión de Protocolo de XMPP). Además, es muy fácil de extender. Su objetivo es proporcionar un marco completo para desarrollar aplicaciones relacionadas con Jabber o scripts en Ruby. (6)

1.3 Zend Framework.

Se trata de un Marco de Trabajo para el desarrollo de aplicaciones y servicios Web con PHP, brinda soluciones para construir sitios web modernos, robustos y seguros. Este Marco de Trabajo está formado por una serie de métodos estáticos y componentes que usarán estos métodos. Los componentes son varios y variados y aunque alguno es posible que no se use nunca, hay otras que puede que se usen

hasta la saciedad, por ejemplo el componente para la base de datos. Como características Zend Framework fundamentales tiene:

Trabaja con Modelo Vista Controlador (MVC).

El Marco de Zend también incluye objetos de las diferentes bases de datos, por lo que es extremadamente simple para consultar su base de datos, sin tener que escribir ninguna consulta SQL. Una solución para el acceso a base de datos que balancea el Mapeador de Objeto Relacional con eficiencia y simplicidad. Cuenta con una completa documentación y test de alta calidad, Soporte avanzado y Robustas clases para autenticación y muchas otras clases útiles para hacerlo tan productivo como sea posible. (7)

Los 51 componentes de Zend Framework conforman un potente y extensible Marco de Trabajo de aplicaciones web al combinarse entre sí. De todos estos Sauxe utiliza solamente los siguientes:

Zend_Cache

Zend_Config

Zend_Controller

Zend_Loader

Zend_Log

Zend_Registry

Zend_Session

Zend_View

A partir de la extensión de algunos componentes de Zend Framework surge ZendExt, desarrollado por el Departamento de Tecnología del Centro de Informatización de la Gestión de Entidades (CEIGE) y la Unidad de Compatibilización Integración y Desarrollo De Software para la Defensa (UCID), con el objetivo de crear un Marco de Trabajo extensible y configurable centrando el desarrollo de las aplicaciones, en la lógica del negocio, en las interfaces de usuario, alejando a los programadores de los detalles arquitectónicos, con soporte para entornos multi-entidad y para una arquitectura de sistema orientada a componentes. (8)

Doctrine es un potente y completo sistema ORM (Mapa de Objeto Relacional) para PHP 5.2.3 que implementa el Lenguaje de Consulta de Datos (DQL) con un DBAL (Capa de Abstracción de Base de Datos) incorporado. Entre otros elementos se tiene 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. (9)

ExtJS es una librería JavaScript ligera y de alto rendimiento, compatible con la mayoría de los navegadores que permite crear páginas e interfaces web dinámicas. (10)

Sauxe utiliza ExtJS en la capa de presentación, por la gran gama de componentes que se pueden reutilizar para agilizar el proceso de desarrollo y mostrarle al usuario una interfaz más amigable y funcional.

La unión de ZendExt, Doctrine y ExtJS dio lugar al Marco de Trabajo Sauxe.

1.4 Marco de Trabajo Sauxe.

Sauxe es un Marco de Trabajo que contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. (8)

Sauxe en su versión 2.0 está formado por los componentes: ZendExt_App, ZendExt_Aspect, ZendExt_ADT, ZendExt_Cache, ZendExt_ExpImp, ZendExt_MVC, ZendExt_Exception, ZendExt_FastResponse, ZendExt_GlobalConcept, ZendExt_IoC, ZendExt_Nomencladores, ZendExt_Trace, ZendExt_Validation, ZendExt_Portal, ZendExt_TransactionManager.

Actualmente Sauxe no cuenta con un sistema de notificación que permita al usuario y al servidor una comunicación en tiempo real por lo que los usuarios no pueden enterarse de los eventos generados por el servidor conllevando esto al atraso de las tareas de las aplicaciones que utilizan este Marco de Trabajo.

1.5 Metodologías de desarrollo

Se conoce como metodología al conjunto de procedimientos, técnicas, herramientas y al soporte documental que ayuda a los desarrolladores a realizar un software. Una metodología define quién debe hacer qué, cuándo y cómo debe hacerlo. (11)

1.5.1Principales exponentes de metodologías de desarrollo

Actualmente las metodologías de desarrollo se clasifican en dos grandes grupos, las denominadas metodologías ágiles y las metodologías tradicionales o pesadas, así conocidas comúnmente. Esta última clasificación ha demostrado ser efectiva y necesaria a la hora de planificar el proceso de desarrollo de software cuando este es para un proyecto muy grande y que requiere una cierta cantidad de tiempo.

Mientras que las metodologías ágiles se utilizan en proyectos de corta duración y de menos envergadura, lo que no quiere decir que renuncien a las prácticas esenciales para asegurar la calidad del producto. Dentro de estas dos clasificaciones de las metodologías las más utilizadas por sus diversas características son:

1. Metodologías ágiles: La programación extrema o extreme Programming (XP), SCRUM y Adaptive Software Development Adaptación de Desarrollo de Software (ASD).

ASD: es un modelo de patrones agiles para desarrollo de software, esta da forma a las fases básicas de la gestión ágil posibilitando un desarrollo concurrente del trabajo de construcción y la gestión del producto. Esta metodología en cada una de sus iteraciones chequea la calidad de acuerdo a los criterios del cliente y los criterios técnicos, controla la funcionalidad desarrolladora y el estado del proyecto.

Posee como características principales un trabajo orientado y guiado por la misión del proyecto basado en la funcionalidad del mismo promoviendo un desarrollo iterativo y a su vez acotado temporalmente. ASD está guiado por los riesgos, lo que le permite reponerse rápidamente de imprevistos, por lo que esto ayuda a que el trabajo que se desarrolla sea tolerante a los cambios, puesto que esta metodología propone que las necesidades del cliente son siempre cambiantes. (12)

Aspectos claves de ASD:

- Emite un conjunto no estandar de artefacto de misión (documentos para ti y para mi) incluyendo una visión del proyecto, una hoja de datos, un perfil de misión del producto y un esquema de su especificación.
- Tiene un ciclo de vida, inherentemente iterativo.
- Posee cajas de tiempo, con ciclos cortos de entrega orientados por riesgo.

ASD como metodología ágil no esta exenta de los riesgos que se corren al planificar el desarrollo de un softaware con metodologías ágiles como la agreción que puede ser provocada por los hackers puesto que la codificacion no esta regida por estandares que no permitan introducir cualquier tipo de código. (13)

SCRUM: Se basa en el principio ágil de desarrollo iterativo e incremental. Esta metodología define un proceso empírico de desarrollo que intenta obtener ventajas respecto a los procesos definidos, mediante la aceptación de la naturaleza caótica del desarrollo de software y la utilización de prácticas tendientes a manejar la impredictibilidad y el riesgo a niveles aceptables.

El funcionamiento de SCRUM está definido sobre su propia estructura la cual está propuesta por diferentes pasos, Al principio del proyecto se define el Product Backlog (Cartera de productos), que contiene todos los requerimientos funcionales y no funcionales que deberá satisfacer el sistema a construir, y a partir de éste se definirán cada una de las iteraciones conocidas como Sprint cuya duración está definida en un periodo de 30 días y en caso excepcionales 60 días. Al finalizar cada Sprint, se realizará un Sprint review para evaluar los artefactos construidos y comentar el planeamiento del próximo Sprint, aquí se evalúan los resultados obtenidos. SCRUM también posee revisiones diarias que las hace el propio equipo en su auto-gestión. Esta metodología se utiliza en situaciones frecuentes en el desarrollo de determinados sistemas de software, producto que es muy factible de utilizar en entornos con requisitos inestables y que requieren rapidez y flexibilidad.

Programación Extrema: Es la primera metodología ágil por lo que se puede expresar que ella dio conciencia al movimiento de metodologías agiles que existen hoy en día. Esta metodología está centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y gran capacidad para enfrentar los cambios. Se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

En que se basa fundamentalmente XP:

- Las pruebas unitarias son visionarias de las posibles fallas que pudieran ocurrir dentro del desarrollo del software.
- La historias de usuarios, estas son unas tarjeticas de papel que el usuario escribe mediante las cuales se obtiene los requisitos funcionales y no funcionales del producto.

- La reutilización de código, que está definida por patrones o estándares para que posibilite la existencia de la flexibilidad ante los cambios.
- La definición muy estructurada y detallada de cada uno de los roles y procesos dentro del ciclo de desarrollo del software.

XP suele diferenciarse de las demás metodologías agiles en un aspecto en particular: el alto nivel de disciplina de las personas que participan en el proyecto, aunque como las demás metodologías trae aparejado desventajas de seguridad y de capacidad en tamaño para proyectos grandes.

2. Metodologías pesadas o tradicionales: Proceso Unificado de Desarrollo -Rational Unified Process (RUP)- y Microsoft Solution Framework (MSF).

Solución de Microsoft Framework (MSF): Es un metamodelo para describir el ciclo de vida de desarrollo del software, esta herramienta brinda un grupo de guías para lograr que la solución de sistemas informáticos pueda ser finalizada exitosamente, rápidamente y reduciendo la cantidad de personas y riesgos. Posee características como adaptabilidad, flexibilidad y escalabilidad. MSF está compuesto por dos modelos principales que son el modelo de equipo y el modelo de procesos y posee tres disciplinas fundamentales administración de proyectos, administración de disposición y administración de riesgos. A lo largo de la aplicación de dicha metodología se ha acumulado experiencia que se encuentra reflejada en unas determinadas guías que no imponen demasiados detalles que se vuelvan imposibles de entender o que solo se apliquen a un número limitado de casos, todo lo contrario posibilitando la aparición de un framework que sea fácil de entender y adoptar.

Proceso unificado de software (RUP): Se conoce que un proceso de desarrollo de software es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software, por lo que se define que el proceso unificado es más que un simple proceso. Esta metodología es una infraestructura flexible de desarrollo, que está basada en componentes que a su vez están interconectados a través de interfaces, posee una arquitectura recomendable y proporciona prácticas recomendadas probadas.

Este proceso unificado de software en conjunto con el lengujae unificado de modelado constituye la metodologia estandar más utilizada para el análisis, implementación y documentación de los sistemas orientados a objetos. RUP posee tres caracteristicas fundamentales:

- Dirigido por casos de uso: el proceso de desarrollo esta guiado por casos de uso ya que estos reflejan en si los requisito funcionales del sistema.
- Centrado en la arquitectura: la arquitectura permite tener diferentes vistas del sistema en contrucción, por lo que describe los elementos del modelo del propio sistema, la arquitectura cienta la bases para producir, el software, económicamente. La arquitectura es la encargada de documentar los procesos, basándose en el Lenguaje de Modelado Unificado Unified Modeling Language (UML).
- Iterativo e incremental: esta característica es la que se encarga de dividir el trabajo en partes mas pequeñas o miniproyectos. Cada miniproyecto es una iteración que resulta en un incremento del producto, pero sin olvidar que en cada miniproyecto debe existir un equilibrio entre la arquitectura y los casos de uso. (11)

1.6 Descripción del modelo de desarrollo

Los modelos de desarrollo definen un conjunto de actividades, acciones, tareas, fundamentos y productos de trabajo que se requieren para desarrollar software de alta calidad. Estos modelos de desarrollo no son perfectos, pero proporcionan una guía útil para el trabajo de la ingeniería del software. (14)

El Modelo de Desarrollo orientado a componentes del CEIGE está orientado a las necesidades y artefactos del mismo, el cual se creó como un híbrido de diversas metodologías de desarrollo, que tiene como características fundamentales:

- Se modela el negocio por procesos, no por casos de usos.
- La ingeniería de requisitos es más clara que en las demás metodologías de desarrollo.
- Es orientada a componentes, posibilitando la independencia de funciones del sistema a la hora de mantener o modificar el sistema funcional.
- Utiliza como lenguajes de modelado BPMN para el negocio y UML para el diseño.

- Se definen claramente los roles que intervienen y las responsabilidades de cada uno de ellos, las actividades de desarrollo que se deben llevar a cabo, bien descritas y que roles la realizan, así como los artefactos y los roles que lo generan. (15)

Roles y responsabilidades:

Roles	Responsabilidades
Jefe de Línea de Desarrollo	Responsable de garantizar los cronogramas y compromisos de la línea Supervisar el proceso de desarrollo Organiza y controla el trabajo de los miembros de su línea Controla los indicadores de eficiencia
Planificador	Mantener actualizado el cronograma Mantener actualizada la plantilla de Capital Humano. Planificar y controlar las tareas de los miembros del equipo, según las prioridades Controlar los horarios de trabajo y distribución de máquinas. Llevar las actas de las reuniones y talleres. Controlar los planes de trabajo Individuales
Arquitecto de Sistema	Que se cumplan las políticas y estándares definidos en la Arquitectura. Las decisiones de integración en el proyecto y la Arquitectura del Sistema. Modera el Taller de Diseño.
Arquitecto de	Construye y actualiza el Modelo de Datos, además responde por el manejo y

Datos	recuperación de la información del mismo
Analista Principal	Dirigir y organizar el trabajo del grupo de analistas de la Línea. Elaborar el Mapa de Procesos de la Línea según los estándares. Participar en la definición y construcción de la Arquitectura de Negocio.
Especialista de Calidad	Revisar, controlar las normas y estándares que establece el grupo de aseguramiento de la calidad incluyendo el proceso de desarrollo. Guiar al grupo de auditoría y revisiones Coordinar el proceso de diseño de casos de prueba. Coordinar las pruebas de aceptación o liberación.
Especialista Funcional	Participar en las sesiones de trabajo para identificar, describir y validar los procesos de negocio y los requisitos de software Validar, desde el punto de vista funcional, los procesos de negocio y requisitos de software Elaborar Casos de Prueba según los estándares establecidos para ello
Analista	Participar en las sesiones de trabajo para identificar, describir y validar los procesos de negocio y los requisitos de software Elaborar la Descripción de Procesos de Negocio, Especificación de Requisitos según los estándares establecidos para ello Participar en el Taller de Diseño
Desarrollador	Diseña y Construye los componentes de software de la línea

Representación de los artefactos por roles:

Roles	Artefactos
Jefe de Línea de Desarrollo	Plan de Iteración Plan de Gestión de Riesgos Plan de Trabajo individual de los integrantes de la Línea
Planificador	Plan de Iteración Plan de Trabajo Individual de los profesionales Definición del cronograma de desarrollo
Arquitecto de Sistema	Plan de Trabajo Individual Diagrama de componentes Prioridad de los componentes Agrupación Requerimientos - Componentes Informe de Integración
Arquitecto de Datos	Plan de Trabajo Individual Modelo de Datos Descripción del Modelo de Dato
Analista Principal	Plan de Trabajo Individual Mapa de Procesos de la Línea
Analista	Plan de Trabajo Individual Modelo de procesos de negocio Descripción de procesos de negocio Modelo Conceptual

	Prototipo de IU Especificación de requisitos.
Especialista de Calidad	Plan de Trabajo Individual Plan de pruebas Casos de Prueba Registro de No Conformidades
Especialista Funcional	Plan de Trabajo Individual Casos de Prueba Validación de Procesos y Requisitos
Desarrollador	Plan de Trabajo Individual Implementación de componentes Descripción de los componentes
Diseñador de LN	Diagrama de Clases Descripción del Diseño de Clases

1.7 Lenguajes de Modelado.

*1.7.1 BPMN (*Notación de Procesos de Gestión Empresarial según sus siglas en ingles*):*

Es una notación que modela los procesos de negocio, basada en diagramas de flujo fácil de entender. Está diseñada para cubrir varios tipos de modelado, permite la creación tanto de segmentos de procesos, como procesos de negocio de comienzo a fin en diferentes niveles de representatividad. BPMN es gráficamente más rico, con menos símbolos fundamentales, pero con más variaciones de éstos, lo que facilita su comprensión por parte de personas no expertas.

- Provee una notación que es fácilmente entendida por todos los usuarios, desde el analista de negocio, el desarrollador técnico y hasta la propia gente del negocio.
- Crea un puente estandarizado para el vacío existente entre el diseño del proceso de negocio y su implementación.
- Asegura que los lenguajes para la ejecución de los procesos de negocio puedan ser visualizados con una notación común. (11)

1.7.2 Lenguaje Unificado de Modelado.

El Lenguaje Unificado de Modelado (UML: Unified Modeling Language) es un lenguaje gráfico para especificar, construir y documentar los artefactos que modelan un sistema. Fue diseñado para ser un lenguaje de modelado de propósito general, que ofrece gran flexibilidad y expresividad y puede utilizarse para especificar la mayoría de los sistemas basados en objetos o en componentes. Igualmente es empleado para modelar aplicaciones de muy diversos dominios de aplicación (telecomunicaciones, comercio, sanidad, entre otros) y plataformas de objetos distribuidos como por ejemplo J2EE, .NET o CORBA. (17)

1.8. Herramienta

1.8.1 Herramientas CASE.

La ingeniería de software asistida por computadora (CASE: Computer Aided Software Engineering: Software Asistida por Computación) ayuda a los ingenieros de software en todas las actividades asociadas a los procesos de software. Las herramientas CASE automatizan las actividades de gestión de proyectos, gestionan todos los productos de los trabajos elaborados a través del proceso y ayudan a los ingenieros en el trabajo de análisis, diseño y codificación. La importancia de su uso radica en permitir reducir la cantidad de esfuerzo que se requiere para producir un producto o para alcanzar un hito en el proceso de desarrollo. Además, contribuyen a la calidad del software ya que proporcionan nuevas formas de observar la información de la ingeniería del software. (18)

Rational Rose Enterprise: Es una herramienta potente en cuanto a modelado visual se refiere está basado en UML y es utilizada mayormente en el análisis y diseño de sistemas orientados a objeto. Este producto se encuentra entre los más completos de la familia de Rational Rose, soporta diferentes patrones y es capaz de modelar ingeniería directa e inversa para lenguajes como Java. Esta herramienta es capaz de hacer revisiones al código de la aplicación y es capaz de generarlo. Permite el modelado UML para bases de datos, dándole la posibilidad de modelar la integración de los requisitos de datos. Posee una interfaz amigable y permite que los diseñadores puedan modelar sus componentes de forma individual y luego unirlos a otros componentes del mismo proyecto. (19) (20) (21)

Enterprise Architect: Es una plataforma avanzada de modelado y diseño, posee una interfaz intuitiva y muy amigable que hace que sea cómoda y fácil su utilización, es una herramienta de modelado orientada a objeto que funciona durante todo el ciclo de vida del producto por lo que podemos asumir que es muy factible para todo el equipo de implementación y desarrollo. Brinda un panel de herramientas bastante completo con diagramas de CU, diagramas de clases y diagramas de secuencia. Actualmente esta herramienta puede ser usada tanto en Windows como en Linux. Es una solución para la visualización, análisis, modelado, pruebas y mantenimiento de una amplia gama de sistemas, software, procesos y arquitecturas. (22)

Visual Paradigm.

Visual Paradigm es una herramienta CASE, que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Ayuda a una más rápida construcción de aplicaciones de calidad, con un menor costo. Permite construir todos los tipos de diagramas necesarios para la documentación del software, así como importar proyectos realizados en Rational Rose. Además, presenta un diseño enfocado al negocio que genera un software de calidad. Permite transformar diagramas Entidad-Relación en esquemas de base de datos y viceversa. Permite realizar ingeniería directa e inversa, obtener el código a partir de diagramas, así como diagramas a partir de un código previamente escrito.(23)

1.9 Patrones de diseño.

Para la construcción del diseño, usualmente se emplean un grupo de patrones o modelos, que no son más que soluciones concretas y técnicas para lograr objetivos específicos y que Identifican: clases,

instancias, roles, colaboraciones y la distribución de responsabilidades. Dentro de los más conocidos y usados se encuentran los GRASP (General Responsibility Assignment Software Patterns, Patrones Generales de Software para Asignación de Responsabilidades), que describen los principios fundamentales de la asignación de responsabilidades a objetos y los GOF (Gang of Four) su traducción al español es “pandilla de los cuatro”, son conocidos por este nombre por ser cuatro los autores que expusieron estos patrones.

Entre las facilidades del uso de estos patrones en las clases definidas en el diseño, se aprovecha la facilidad de delegar las responsabilidades a una clase especializada y así evitar en una clase un sobrecargo de relaciones y métodos que se pueden acoplar en terceras. El grupo de GOF se propone unos principios fundamentales de diseño subyacentes a los patrones de diseño que permiten crear aplicaciones más flexibles y robustas; se clasifican los patrones en 3 grandes categorías basadas en su propósito:

- Creación, abstraen el proceso de obtención de instancias.
- Estructurales, se ocupan de cómo clases y objetos son utilizados para componer estructuras de mayor tamaño.
- Comportamiento, atañen a los algoritmos y a la asignación de responsabilidades entre objetos.

Fachada Simplifica los accesos a un conjunto de objetos relacionados, funciona como intermediario entre uno y otro grupo objetos, se utiliza para comunicar estos dos grupos.

Tipo: Estructura, a nivel de objetos.

Propósito: Simplificar el acceso a un conjunto de clases o interfaces. Proporcionar una interfaz unificada de alto nivel que representa a todo un subsistema facilitando su uso. La fachada satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas.

Motivación: Reducir la dependencia entre clases

Fachada ofrece un punto de acceso al resto de clases si estas cambian o se sustituyen por otras solo hay que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones cliente, No oculta las clases sino que ofrece una forma más sencilla de acceder a ellas, en los casos en que se requiere se puede acceder directamente a ellas.

Modelo Vista Controlador (MVC).

El patrón de arquitectura conocido como Modelo-Vista-Controlador (MVC), separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario; es decir separa en tres capas diferentes los datos de una aplicación, la interfaz de usuario, y la lógica de control:

Modelo: Esta capa administra el comportamiento y los datos del dominio de la aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

Vista: Esta capa maneja la visualización de la información, es decir que presenta el modelo en un formato adecuado para interactuar, que usualmente es la interfaz de usuario.

Controlador: Esta capa controla el flujo de datos entre la vista y el modelo; es decir que responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases.

Esta separación permite construir y probar el modelo, independientemente de la representación visual. Este patrón se utiliza frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el modelo es el Sistema de Gestión de Base de Datos y el controlador representa la lógica de negocio.

El patrón de arquitectura MVC tiene flexibilidad para cambiar las vistas y los controladores adaptándose al cambio.

Singleton

El Singleton es quizás el más sencillo de los patrones que se presentan en el catálogo del GoF. Es también uno de los patrones más conocidos y utilizados. Su propósito es asegurar que sólo exista una instancia de una clase. El patrón Singleton garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ésta instancia.

El funcionamiento de este patrón es muy sencillo y podría reducirse a los siguientes conceptos:

1. Ocultar el constructor de la clase Singleton, para que los clientes no puedan crear instancias.
2. Declarar en la clase Singleton una variable miembro privada que contenga la referencia a la instancia única que queremos gestionar.

3. Proveer en la clase Singleton una función o propiedad que brinde acceso a la única instancia gestionada por el Singleton. Los clientes acceden a la instancia a través de esta función o propiedad.

Observador

El patrón Observador es un patrón fácil de entender, y fácil de ver cuando se necesita. Está clasificado como un patrón de comportamiento.

El contexto de este patrón es cuando tenemos varios objetos observadores y un objeto observado por los observadores**.** Los observadores necesitan saber cuando se produce un cambio en el objeto observado. El primer planteamiento que se puede pensar, es que haya un proceso que se encargara de que los observadores vayan haciendo peticiones periódicamente del estado del objeto observado, para así detectar cuando sucede el cambio.

El problema surge cuando el intervalo entre petición es muy corto, o hay demasiados objetos observadores haciendo peticiones al objeto observado, o es necesario que el observador sea notificado inmediatamente después del cambio del objeto observado.

La solución que se aplicaría usando el patrón Observador, trata de que los objetos observadores se añadan a una lista de objetos, y el objeto observado notificará el cambio a todos los objetos de esta lista cuando se produzca el cambio. De esta manera, se produce un cambio de papeles: el observador que tenía la tarea de estar pendiente del cambio, ahora está a la espera de que el observador le avise. Vamos a intentar detallarlo:

- El objeto observador necesita ser notificado de los cambios de un objeto observado.
- El objeto observador se añade a la lista de objetos a notificar, de ese objeto observado.
- En el objeto observado se produce un cambio.
- El objeto observado recorre la lista de objetos observadores apuntados, y les va notificando el cambio.

1.10 Fundamentación de las herramientas, modelo y lenguajes.

Todas las herramientas, modelos y lenguajes relacionados en el presente trabajo, fueron seleccionados por el Centro de Informatización de la Gestión de Entidades (CEIGE). Para la modelación del sistema de Alertas y Avisos se utilizará como herramienta de modelado el Visual Paradigm, que aunque es propietaria, la UCI posee una licencia para el trabajo con dicha herramienta. Como Modelo a emplear para la construcción del mismo se utiliza el Modelo de Desarrollo orientado a componentes del CEIGE, para que cada uno de los equipos de desarrollo posean un modelo estandarizado, así como una definición clara y precisa de las responsabilidades de cada uno de los roles que se ven involucrados en el desarrollo de la solución y como lenguaje de modelado se empleará BPMN para el negocio y UML para el diseño.

CONCLUSIONES PARCIALES

Se realizó un estudio del estado del arte de los Marcos de Trabajo que utilizan sistemas de notificación en tiempo real, logrando sintetizar sus objetivos, características, ventajas y desventajas de su desarrollo, situación que permite comprender su importancia y la necesidad de desarrollar un sistema de esta índole para el Marco de Trabajo Sauxe con sus propias características, ya que luego de realizada la investigación se llego a la conclusión que los componentes de Alertas y Avisos que están implementados no se adaptan a las necesidades de Sauxe.

Capítulo 2: Modelado de negocio y sistema

INTRODUCCIÓN

En este capítulo se presenta la propuesta de solución de la investigación. Se desarrollan varias actividades y artefactos que permiten darle cumplimiento al problema planteado, entender el dominio de la información del problema, definir las funciones que debe realizar el componente y dividir en forma jerárquica los modelos que representan la información y las funciones.

2.1 Análisis del entorno del problema

El componente Alertas y Avisos, debe englobar operaciones que complementen la realización de otros procesos, servir de soporte funcional y simplificar la ejecución de otros eventos. En la propuesta de solución, se agrupan este conjunto de operaciones en un componente con el objetivo de establecer comunicación entre el cliente y el servidor, brindando la posibilidad de tener la información de los eventos generados por el servidor en tiempo real dando la posibilidad de agilizar el trabajo de quienes precisen de estos datos.

Si se parte de que un proceso del negocio es el conjunto estructurado de las actividades que han sido diseñadas para producir un resultado específico para la organización, sus inversores o sus clientes. El análisis anterior indica que es posible identificar los procesos a informatizar, así como los involucrados en cada una de las operaciones. Esto muestra la necesidad de realizar un modelo conceptual y un diagrama de procesos para su posterior implementación. A continuación se muestran los procesos que fueron identificados, con una secuencia definida de acciones, su descripción y modelado.

2.2. Modelación de los procesos de negocio

2.2.1 Descripción del proceso

La modelación de proceso de negocio permite realizar una exploración del dominio del problema, con el fin de lograr comprensión por parte del equipo de desarrollo de los procesos que se realizan actualmente en la entidad y la relación que existe entre estos. De esta forma se van determinando necesidades

operacionales, así como restricciones que presenta la entidad, obteniéndose finalmente un entendimiento del negocio para dar paso a la fase inicial del sistema, lográndose a través de los objetivos siguientes:

- Realizar un estudio de los procesos existentes con el fin de contribuir con el principio de reutilización.
- Detallar las características del negocio a través de la descripción de los procesos.
- Verificar que se haya realizado un buen análisis del negocio. (24)

De acuerdo a lo concebido en el Modelo de desarrollo orientado a componentes utilizado, se muestra a continuación el modelo de procesos perteneciente Aletas y Avisos utilizando para su creación la notación de modelado BPMN:

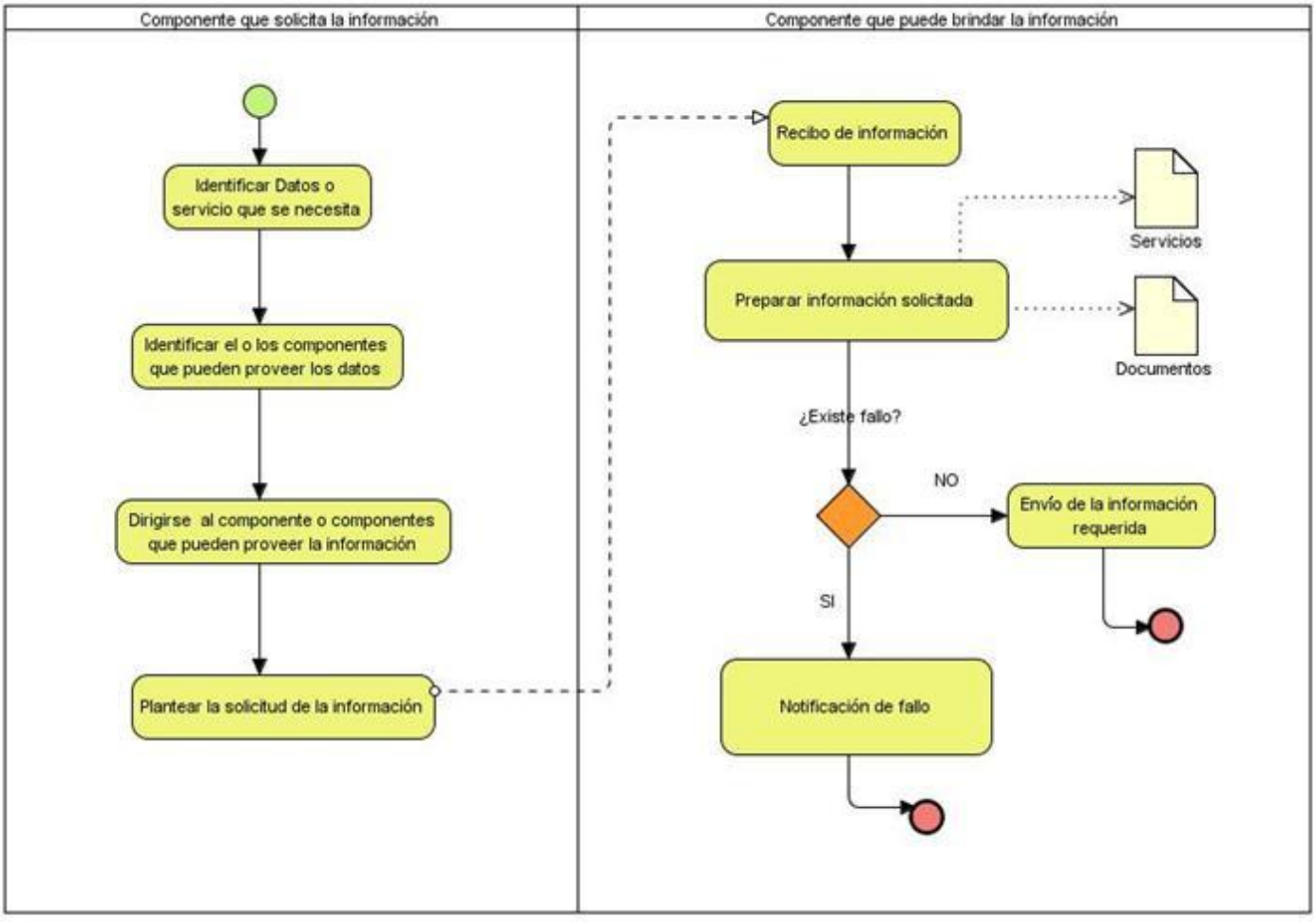


Figura 1. Diagrama de procesos Notificar Alerta

Tabla 1 Descripción de Procesos del negocio

Objetivo	Lograr el intercambio de información entre los clientes y el servidor del Marco de Trabajo Sauxe.
Evento(s) que lo genera(n)	Necesidad de intercambiar información.
Pre condiciones	Necesidad de información.
Marco legal	
Clientes internos	Subsistemas del Marco de Trabajo Sauxe.
Clientes externos	Todos los clientes del Marco de Trabajo Sauxe.
Entradas	
Flujo de eventos	
Flujo básico Notificar alerta	
1.	Identificar la carencia de información
2.	Identificar el componente o los componentes que compensan la información.
3.	Solicitar el servicio o la información de ese componente.
4.	El componente que tiene la información requerida procede a implementar el servicio o a preparar la información.
5.	El componente que tiene la información requerida procede enviar la información o a notificar de la terminación de la implementación del servicio requerido.
6.	Procesamiento de la información por parte de quien la solicita.
Pos-condiciones	
1.	El envió de la información a ocurrido satisfactoriamente.
Salidas	
1. Flujo de información Coherente y ordenada.	

Flujos paralelos	
1.	
2.	
Pos-condiciones	
1.	
Salidas	
1.	
Flujos alternos	
1. a Fallo en el servicio.	
1.	Se detecta un mal funcionamiento de uno de los componentes proveedores del servicio.
2.	Enviar notificación al cliente de la imposibilidad de ejecutar la operación requerida.

Pos-condiciones
Salidas
Asuntos pendientes
Posibles mejoras al proceso.

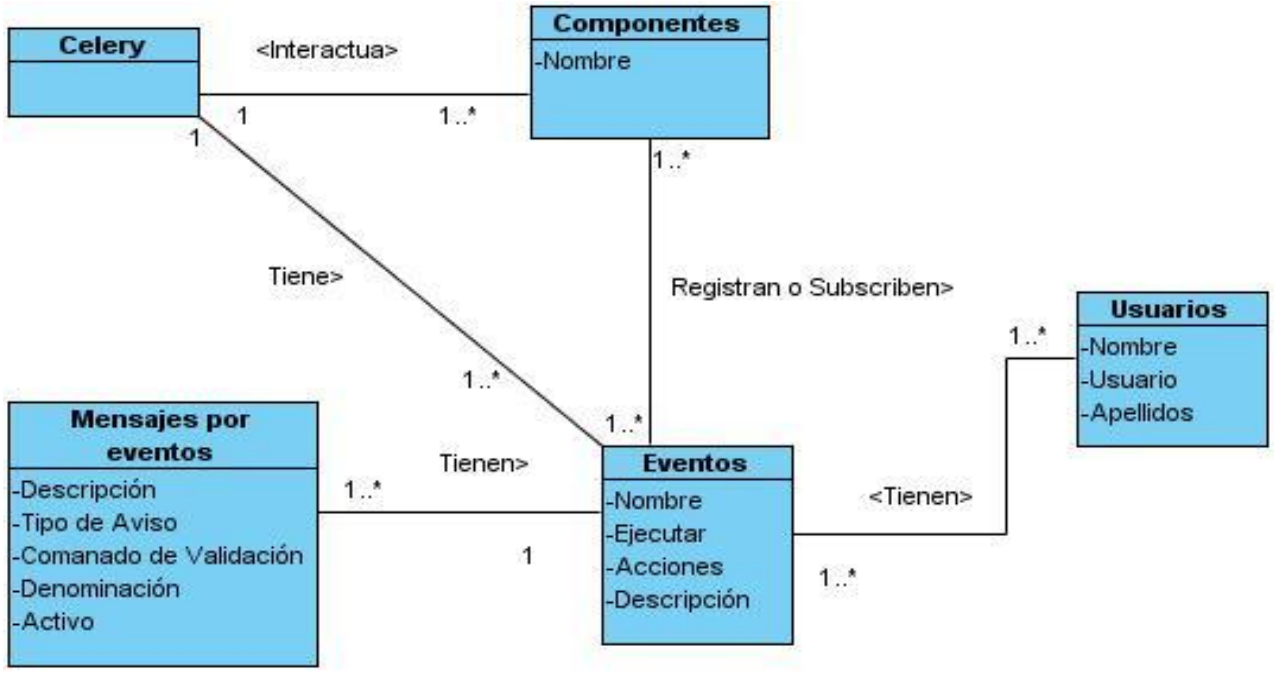


Figura 2. Modelo Conceptual

2.2 Conceptos del modelo conceptual y sus descripciones.

Tabla 2 Eventos

Descripción	Es una alerta o un Incidente provocado por cada componente					
Atributos						
Nombre	**Descripción**	**Tipo**	**¿Puede ser nulo?**	**¿Es único?**	**Restricciones**	
Nombre	Este atributo especifica el nombre del evento registrado.	String	no	si	Letras y números	Caracteres especiales (/*-+!”·$%&)
Acciones	Este atributo especifica el	String	no	si	Letras, números	

	servicio que consume este subsistema			y Caracteres especiales (/*-+!”·$%&)	
Ejecutar	Este atributo especifica El momento en el que se ejecutara el aviso puede ser en el momento o después de ocurrido el evento.	String	no	si	Letras y números	Caracteres especiales (/*-+!”·$%&)
Descripción	Este atributo especifica las características del evento.	String	no	si	Letras y números	Caracteres especiales (/*-+!”·$%&)

Tabla 3 Registrar Eventos

Descripción	Se registra en Celery el evento provocado por cada componente

Tabla 4 Subscribir Eventos

Descripción	Los componentes se subscriben a algún evento o eventos seleccionados para ser notificados posteriormente.

Tabla 5 Notificar Eventos

Descripción	Permite notificar a todos los componentes subscritos de algún evento ocurrido previamente.

Tabla 6 Celery

Descripción	Permite a los componentes registrar y subscribir eventos

Tabla 7 Componentes

Descripción	Son todos los componentes integrados al Marco de Trabajo Sauxe				
Atributos					
Nombre	**Descripción**	**Tipo**	**¿Puede ser nulo?**	**¿Es único?**	**Restricciones**
Nombre	Este atributo especifica el nombre del componente	String	no	si	Letras, números y Caracteres especiales (/*-+!”·$%&)

Tabla 8 Mensajes

Descripción	Son los mensajes emitidos por Celery tras la aparición de algún evento.				
Atributos					
Nombre	**Descripción**	**Tipo**	**¿Puede ser nulo?**	**¿Es único?**	**Restricciones**
Datos	Este atributo especifica los datos que serán mostrados en el mensaje de alerta	Array	no	si	Letras, números y Caracteres especiales (/*-+!”·$%&)

Tabla 9 Mensaje por Eventos

Descripción	Es una notificación que se lanza tras la ocurrencia de un evento.				
Atributos					
Nombre	**Descripción**	**Tipo**	**¿Puede ser nulo?**	**¿Es único?**	**Restricciones**
Descripción	Este atributo especifica el mensaje que desea que se le muestre al	String	no	si	Letras y números Caracteres especiales (/*-+!”·$%&)

	configurar la alerta				
Tipo de aviso	Este atributo especifica El tipo de aviso que se lanzara que puede ser de información u otros	String	no	si	
Comando de validación	Este atributo especifica La funcionalidad que utiliza este servicio.	String	no	si	
Denominación	Este atributo especifica El nombre del aviso	Entero	no	si	Letras y Números Caracteres especiales (/*-+!”·$%&)
Activo	Este atributo especifica si la alerta esta activa o no.	String	no	si	

Tabla 10 Usuarios

Descripción	Son los usuarios a los que se les notificará tras la aparición de algún evento.				
Atributos					
Nombre	**Descripción**	**Tipo**	**¿Puede ser nulo?**	**¿Es único?**	**Restricciones**
Nombre	Este atributo especifica el nombre del usuario	String	no	no	Letras, números y Caracteres especiales (/*-+!”·$%&)
Usuario	Este atributo especifica el usuario UCI del mismo.	String	no	si	Letras, Caracteres especiales (@)
Apellidos	Este atributo especifica los apellidos del usuario	String	no	no	Letras

2.3 Especificación de requisitos.

Para dar cumplimiento al objetivo planteado y conocidos los conceptos que rodean al objeto de estudio, se deben definir una serie de requisitos funcionales que sean capaces de resolver el problema existente.

Técnicas utilizadas en las actividades de la ingeniería de requisitos.

Entrevistas y Cuestionarios

Las entrevistas y cuestionarios se emplean para reunir información proveniente de personas o de grupos. Durante la entrevista, el analista conversa con el encuestado; el cuestionario consiste en una serie de preguntas relacionadas con varios aspectos de un sistema. Por lo común, los encuestados son usuarios de los sistemas existentes o usuarios en potencia del sistema propuesto. En algunos casos, son gerentes o empleados que proporcionan datos para el sistema propuesto o que serán afectados por él. El éxito de esta técnica, depende de la habilidad del entrevistador y de su preparación para la misma.

Sistemas existentes

Esta técnica consiste en analizar distintos sistemas ya desarrollados que están relacionados con el sistema a ser construido. Por un lado, se pueden analizar las interfaces de usuario, observando el tipo de información que se maneja y cómo es manejada, por otro lado también es útil analizar las distintas salidas que los sistemas producen (listados, consultas, etc.), porque siempre pueden surgir nuevas ideas sobre la base de estas.

2.3.1Requisitos funcionales del sistema.

Gestionar Avisos

RF1 Adicionar avisos.
RF2 Eliminar avisos.
RF3 Modificar avisos.
RF4 Listar Avisos

Gestionar Destinatarios.

RF5 Adicionar Destinatario.
RF6 Eliminar Destinatario.
RF7 Modificar Destinatario.
RF8 Listar Nombre de grupo o contacto
RF9 Listar Contactos

RF10 Listar Direcciones de contacto

RF11 Buscar Contactos

Gestionar Grupo.

RF12 Adicionar grupo.

RF13 Eliminar grupo.

RF14 Importar desde estructura.

RF15 Listar Nombre de grupo

RF16 Listar Contactos

RF17 Listar Direcciones de contacto

Gestionar Eventos

RF18 Adicionar Eventos.

RF19 Eliminar Eventos.

RF20 Modificar Eventos

RF21 Listar Eventos

2.3.2 Especificaciones de los requisitos.

RF1. Requisito funcional Gestionar Aviso.

Este requisito gestiona las posibles notificaciones de los eventos que se pueden generar en el servidor en tiempo real, permite al usuario subscribirse a ellas para luego ser notificado.

Especificación de requisito Adicionar Aviso

Descripción textual del requisito

Tabla 11 Adicionar Aviso

Precondiciones	Solo el administrador tiene los permisos para ejecutar esta acción.
Flujo de eventos	
Flujo básico	
1	Seleccionar la interfaz Gestionar Aviso.
2	Seleccionar en la interfaz Eventos el evento al que desea adicionarle el aviso

3	Seleccionar la opción Adicionar de la interfaz Aviso.
4	Entrar los datos del aviso que adicionará.
5	Presionar el botón Aceptar.
6	El sistema confirma que se adicionó correctamente el aviso.
Pos-condiciones	
1	Se ha adicionado un nuevo aviso.
Flujos alternativos 4.a El usuario inserta datos erróneos	
1	El sistema señala los datos erróneos y permite corregirlos.
2	El administrador corrige los datos.
3	Volver al paso 4 del flujo básico.
Flujo alternativo 4.b Información incompleta	
1.	El sistema señala los datos vacíos y permite corregirlos.
2.	El usuario corrige los datos.
3.	Volver al paso 4 del flujo básico.
Pos-condiciones	
No se adiciona la información.	
Flujo alternativo *.a El usuario cancela la acción	
1	Concluye el requisito.
Pos-condiciones	
1	No se adiciona la información.
Validaciones	
Se validan los datos según lo establecido en el Modelo conceptual(Anexo)	
Relaciones	**Requisitos Incluidos**
	Extensiones N/A.
Conceptos	**Procesos**
	Denominación
	Descripción
	Medios de aviso

	Tipo de aviso Activo Comando de validación
Requisitos especiales	
Asuntos pendientes	Posibles mejoras al requisito.

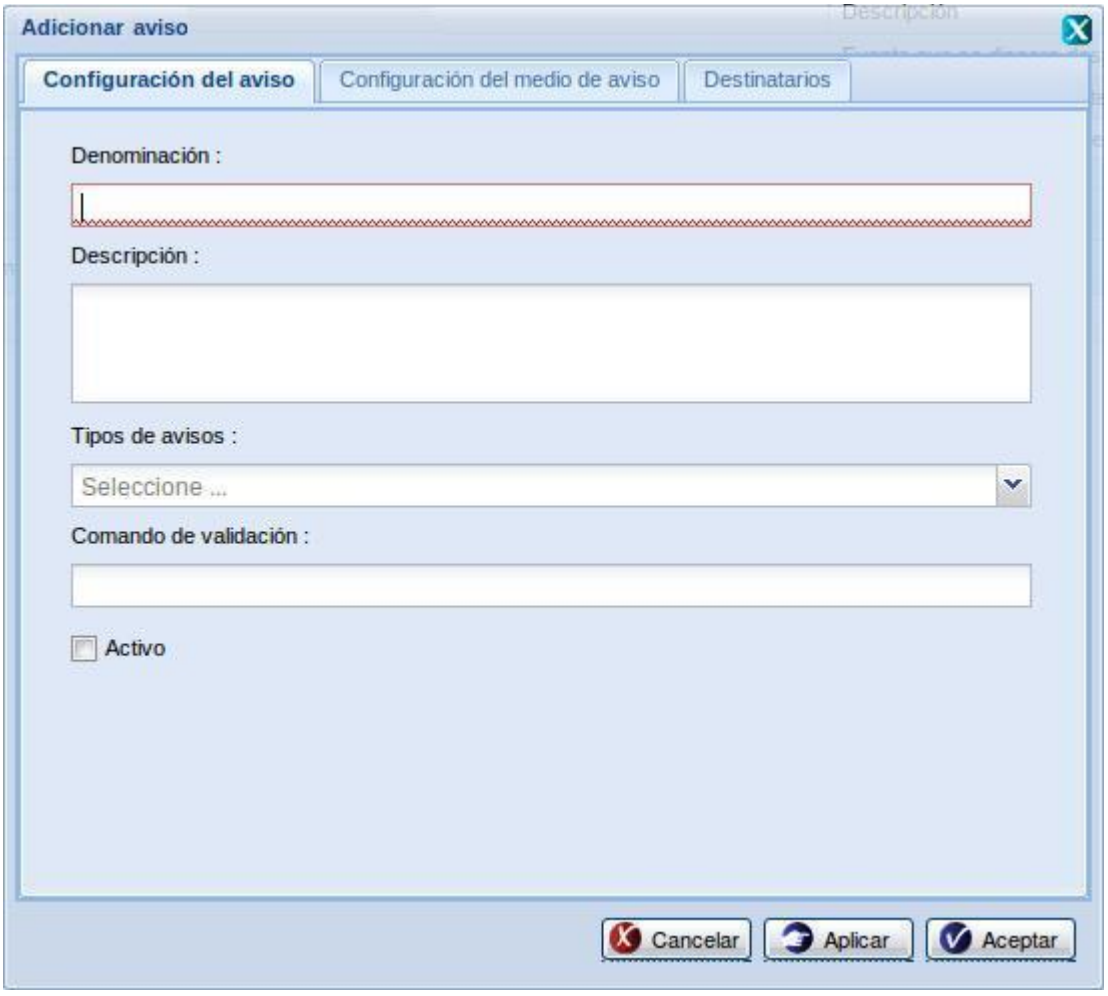


Figura 3. Adicionar Aviso

Especificación de requisito Adicionar Destinatario

RF2. Requisito funcional Gestionar Destinatario.

Este requisito gestiona los posibles usuarios de los avisos.

Descripción textual del requisito

Tabla 12 Adicionar Destinatario

Precondiciones	Solo el administrador tiene los permisos para ejecutar esta acción.
Flujo de eventos	
Flujo básico	
1	Seleccionar la interfaz Gestionar Aviso.
2	Seleccionar la opción Adicionar de la interfaz Aviso.
3	Seleccionar la opción Adicionar de la interfaz Destinatario.
4	Entrar los datos requeridos.
5	Presionar el botón Aceptar.
7	El sistema confirma que se adicionó correctamente el Destinatario.
Pos-condiciones	
2	Se ha adicionado un nuevo Destinatario.
Flujos alternativos 4.a Información incompleta	
4.	El sistema señala los datos vacíos y permite corregirlos.
5.	El usuario corrige los datos.
6.	Volver al paso 4 del flujo básico.
Flujo alternativo 4.b El usuario inserta datos erróneos	
1.	El sistema señala los datos erróneos y permite corregirlos.
2.	El administrador corrige los datos.
3.	Volver al paso 4 del flujo básico.
Pos-condiciones	

1. No se insertan los datos		
Flujo alternativo *.a El usuario cancela la acción		
1 Concluye el requisito.		
Pos-condiciones		
1. No se insertan los datos		
Validaciones		
Se validan los datos según lo establecido en el Modelo conceptual(Anexo)		
Relaciones	**Requisitos**	Listar Contactos
	Incluidos	Listar Direcciones
		Buscar Contactos
	Extensiones	N/A.
Conceptos	**Procesos**	Contactos
		Dirección
		Tipo
Requisitos especiales		
Asuntos pendientes	Posibles mejoras al requisito.	

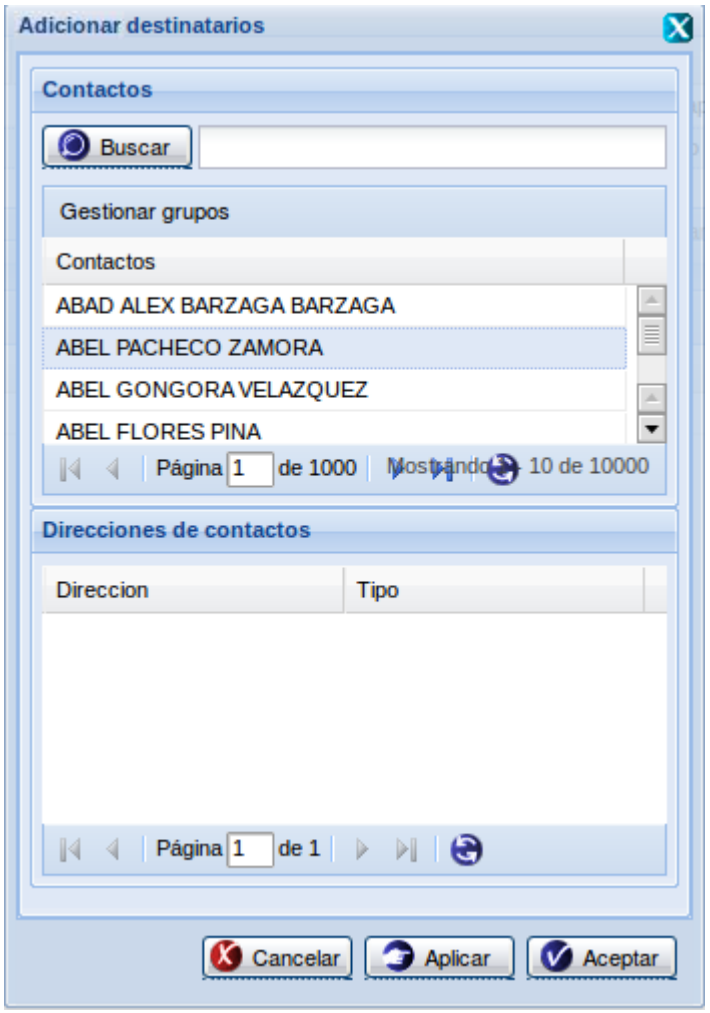


Figura 4. Adicionar destinatario

RF3 Requisito funcional Gestionar Grupo.

En el requisito se gestionan los tipos de usuarios que se subscribirán a los eventos para ser notificados ya sean usuarios individuales o grupos de usuarios.

Especificación de requisito Adicionar Grupo

Descripción textual del requisito

Tabla 13 Adicionar Grupo

Precondiciones	Solo el administrador tiene los permisos para ejecutar esta acción.
Flujo de eventos	
Flujo básico	
1	Seleccionar la interfaz Gestionar Aviso.
2	Seleccionar la opción Adicionar de la interfaz Aviso.
3	Seleccionar la opción Adicionar de la interfaz Destinatario.
4	Seleccionar la opción Gestionar grupo.
5	Seleccionar la opción adicionar de la interfaz grupo
6	Entrar los datos requeridos.
7	Seleccionar los contactos que desea adicionar.
8	Presionar el botón Aceptar.
9	El sistema confirma que se adicionó correctamente el grupo.
Pos-condiciones	
1	Se ha adicionado un nuevo grupo.
Flujos alternativos 6.a Información incompleta	
1	El sistema señala los datos vacíos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 6 del flujo básico.
Flujo alternativo 6.b El usuario inserta datos erróneos	
1.	El sistema señala los datos erróneos y permite corregirlos.
2.	El administrador corrige los datos.
3.	Volver al paso 6 del flujo básico.
Pos-condiciones	
1.	No se adiciona el grupo

Flujo alternativo *.a El usuario cancela la acción		
1 Concluye el requisito.		
Pos-condiciones		
Validaciones		
Se validan los datos según lo establecido en el Modelo conceptual(Anexo)		
Relaciones	**Requisitos Incluidos**	Listar Nombre de grupo Listar Contacto Listar direcciones
	Extensiones	N/A.
Conceptos	**Procesos**	Nombre Nombre del grupo Direcciones
Requisitos especiales		
Asuntos pendientes	Posibles mejoras al requisito.	

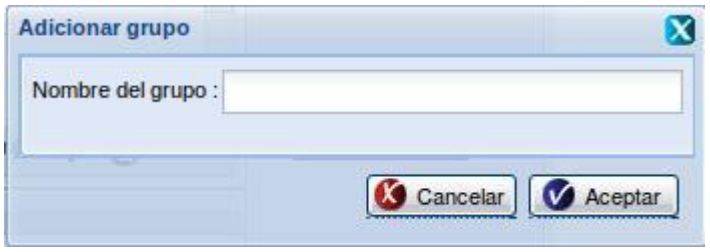


Figura 5. Adicionar grupo

RF4 Requisito funcional Gestionar Eventos

Especificación de requisito Adicionar Eventos

Descripción textual del requisito

Tabla 14 Adicionar Eventos

Precondiciones	Solo el administrador tiene los permisos para ejecutar esta acción.
Flujo de eventos	
Flujo básico	
1	Seleccionar la interfaz Gestionar Aviso.
2	Seleccionar la opción Adicionar de la interfaz Eventos.
3	Entrar los datos del evento que se adicionará.
4	Presionar el botón Aceptar.
5	El sistema confirma que se adicionó correctamente el evento.
Pos-condiciones	
2.	Se ha adicionado un nuevo evento.
Flujos alternativos	
Flujo alternativo 3.a Información errónea	
1	El sistema señala los datos erróneos y permite corregirlos.
2	El administrador corrige los datos.
3	Volver al paso 3 del flujo básico.
Pos-condiciones	
1	No se adiciona el evento.
Flujo alternativo 3.b Información incompleta	
1	El sistema señala los datos vacíos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 3 del flujo básico.
Pos-condiciones	

1	No se adiciona el evento.
Flujo alternativo *.a El administrador cancela la acción	
1	Concluye el requisito.
Pos-condiciones	
1	No se adiciona el evento.
Validaciones	
Se validan los datos según lo establecido en el Modelo conceptual(Anexo)	

Relaciones	**Requisitos Incluidos**	.
	Extensiones	N/A.
Conceptos	**Proceso**	Nombre del evento Ejecutar Acciones Descripción
Requisitos especiales	N/A.	
Asuntos pendientes	N/A.	

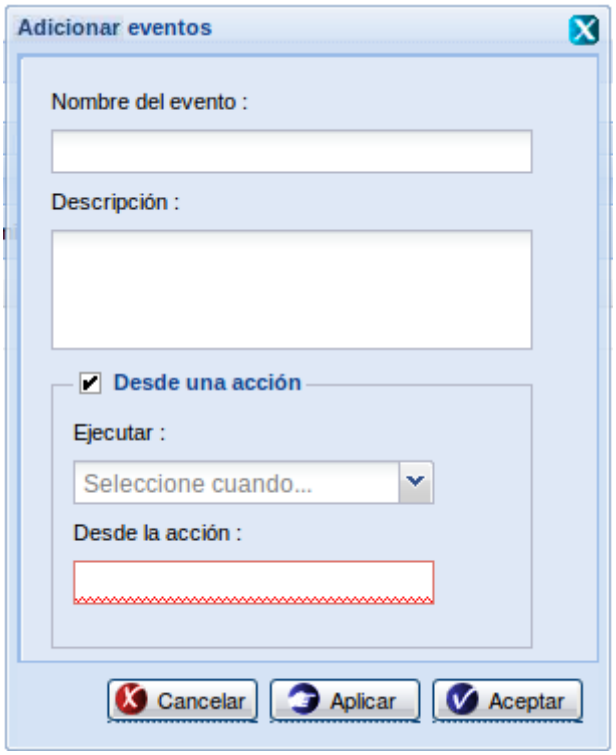


Figura 6. Adicionar eventos

2.4 Requisitos no Funcionales

Usabilidad.

El sistema propuesto estará estructurado de forma sencilla, de manera que pueda ser usado por personas con conocimientos mínimos en el trabajo con una computadora.

Rendimiento.

Los tiempos de respuesta y velocidad de procesamiento de la información serán rápidos, no mayores de 5 segundos para las actualizaciones y 20 para las recuperaciones.

Seguridad.

El sistema debe garantizar el control en el acceso, utilizando la autentificación y autorización mediante contraseñas. Además debe garantizar la protección contra acciones no autorizadas o que puedan afectar la integridad de los datos. La atención al sistema incluyendo el mantenimiento de las bases de datos, así como la salva de la información, se realizará de forma centralizada por el administrador.

Requerimiento de software.

Para el cliente:

- Navegador Mozilla Firefox 3.0 o superior.
- Sistema operativo Windows XP o superior o Linux en cualquiera de sus distribuciones.

Para el servidor:

- Sistema operativo Linux en cualquiera de sus distribuciones.
- Un servidor Apache 2.0 o superior con módulo PHP 5.0 disponible. Este debe estar configurado con la extensión “pgsql” incluida.
- Un servidor de base de datos PostgreSQL 8.3 o superior.

Requerimientos de hardware.

Para el cliente:

- Requerimientos mínimos: Procesador Pentium III a 1GHz con 256Mb de memoria RAM.
- Tarjeta de red.

Para el servidor:

- Requerimientos mínimos: Procesador Pentium IV a 2GHz de velocidad de procesamiento y 1Gb de memoria RAM.
- Al menos 40Gb de espacio libre en disco duro.
- Tarjeta de red.

2.5 Diagramas de clases del diseño.

En el diseño se modela el sistema y se encuentra su forma, incluyendo la arquitectura, para que soporte todos los requisitos y otras restricciones relacionadas con el entorno de la implementación, tiene impacto en el sistema a desarrollar. Además, el modelo de diseño sirve de abstracción de la implementación del sistema y es utilizado como entrada fundamental de las actividades de implementación.

Las siguientes ilustraciones muestran los diagramas de clases del diseño y las relaciones entre estas.

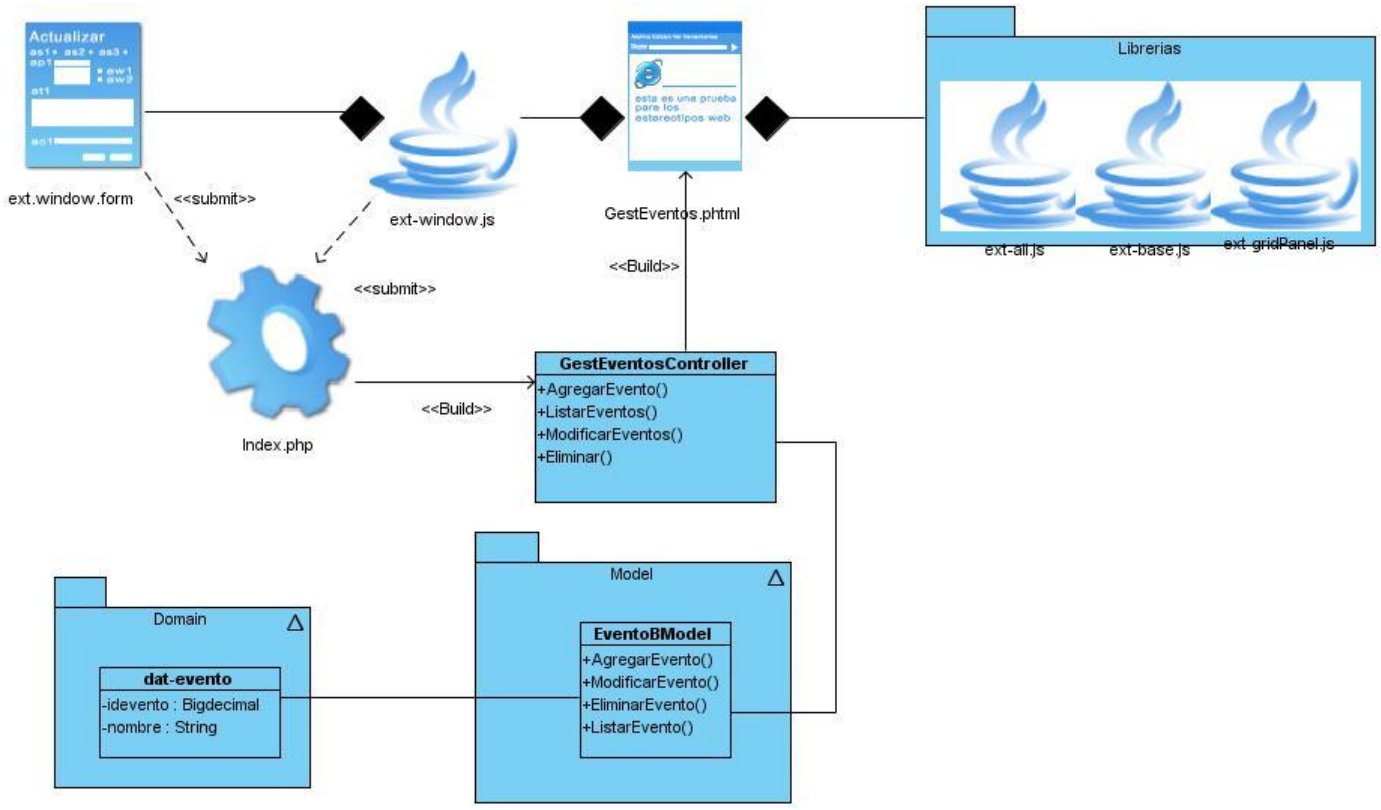


Figura 7 Diagrama de clase Gestionar Eventos.

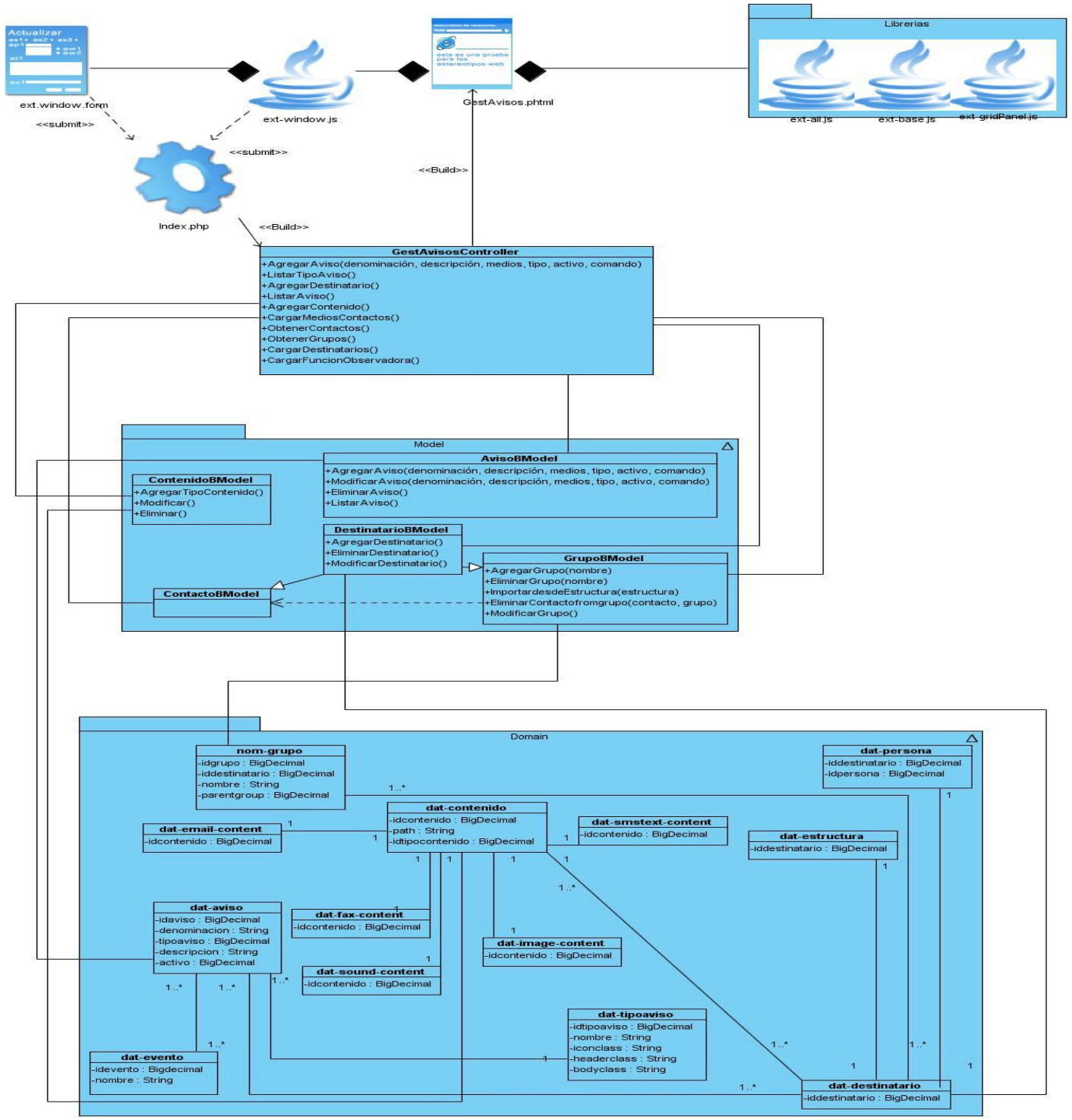


Figura 8. Diagrama de clase Gestionar Avisos.

2.5.1 Descripciones de las clases de diseño.

Tabla 15 Descripción de la clase GestAvisosController

Nombre: GestAvisosController	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	AgregarAviso(denominación,descripción,medios,tipo,activo,comando)
Descripción:	Permite agregar un aviso al sistema utilizando los métodos AgregarDestinatario(),AgregarContenido().
Nombre:	AgregarDestinatario()
Descripción:	Permite agregar un destinatario al sistema con el objetivo de tener registrados los usuarios a los que se le enviara la notificación.
Nombre:	ObtenerGrupo()
Descripción:	Permite obtener los grupos insertados previamente por el usuario para cargarlos en la interfaz de usuario.
Nombre:	ListarTipoAviso()
Descripción:	Permite listar en la interfaz de usuario los tipos de aviso existentes.
Nombre:	CargarMediosContactos()
Descripción:	Esta responsabilidad carga los datos medio del aviso e id de la persona a la que se le notificara del aviso con el fin de saber los medios específicos por los que se le puede notificar del evento ocurrido.
Nombre:	CargarDestinatarios()
Descripción:	Permite cargar en la interfaz de usuario destinatario los nombres de los contactos o grupos que serán notificados.
Nombre:	ListarAviso()
Descripción:	La responsabilidad permite listar los avisos agregados en la interfaz de usuario.
Nombre:	ObtenerContactos()
Descripción:	Permite obtener los contactos que se mostrarán en la interfaz destinatario
Nombre:	AgregarContenido()

Descripción:	Esta responsabilidad permite agregar el contenido que llevará el medio del aviso.
Nombre:	CargarFuncionObservadora()
Descripción:	Esta funcionalidad permite cargar una función observadora que se encargará de monitorear el aviso.

2.5.2 Descripciones de las clases de diseño.

Tabla 16 Descripción de la clase AvisoBModel

Nombre: AvisoBModel	
Tipo de clase:	
Para cada responsabilidad:	
Nombre:	AgregarAviso(denominación,descripción,medios,tipo,activo,comando)
Descripción:	Esta responsabilidad permite adicionar un aviso al sistema enviando los datos obtenidos a la base de datos utilizando la clase dat-aviso.
Nombre:	ModificarAviso()
Descripción:	Esta responsabilidad permite modificar un aviso del sistema enviando los datos obtenidos a la base de datos utilizando la clase dat-aviso.
Nombre:	EliminarAviso ()
Descripción:	Esta responsabilidad permite eliminar un aviso del sistema eliminando los datos de la base de datos utilizando la clase dat-aviso.
Nombre:	ListarAviso()
Descripción:	Esta responsabilidad permite listar los avisos en el sistema.

2.5.3 Descripciones de las clases de diseño.

Tabla 17 Descripción de la clase EventosBModel

Nombre: EventoBModel
Tipo de clase:
Para cada responsabilidad:

Nombre:	AgregarEventos()
Descripción:	Esta responsabilidad permite adicionar un evento a la base de datos utilizando la clase dat-evento.
Nombre:	ModificarEventos()
Descripción:	Esta responsabilidad permite modificar un evento a la base de datos utilizando la clase dat-evento.
Nombre:	EliminarEventos()
Descripción:	Esta responsabilidad permite eliminar un evento de la base de datos utilizando la clase dat-evento.
Nombre:	listarEventos()
Descripción:	Esta responsabilidad permite listar los eventos del sistema**.**

2.5.4 Descripciones de las clases de diseño.

Tabla 18 Descripción de la clase GrupoBModel

Nombre: GrupoBModel	
Tipo de clase:	
Para cada responsabilidad:	
Nombre:	AgregarGrupo(nombre)
Descripción:	Esta responsabilidad permite agregar un grupo a la base de datos utilizando la clase nom-grupo.
Nombre:	EliminarGrupo()
Descripción:	Esta responsabilidad permite eliminar un grupo de la base de datos utilizando la clase nom-grupo.
Nombre:	ImportardesdeEstructura(estructura)
Descripción:	Esta responsabilidad permite importar una estructura definida previamente.
Nombre:	EliminarContactofromgrupo(grupo,contacto)
Descripción:	Esta responsabilidad permite eliminar un contacto de un grupo definido previamente.
Nombre:	AgregarContactoGrupo(grupo,contacto)

Descripción:	Esta responsabilidad permite adicionar un contacto a un grupo.
Nombre:	ModificarGrupo()
Descripción:	Esta responsabilidad permite modificar un grupo de la base de datos utilizando la clase nom-grupo.

2.5.5 Descripciones de las clases de diseño.

Tabla 19 Descripción de la clase DestinatarioBModel

Nombre: DestinatarioBModel	
Tipo de clase:	
Para cada responsabilidad:	
Nombre:	AgregarDestinatario(nombre)
Descripción:	Esta responsabilidad permite adicionar un destinatario a la base de datos utilizando la clase dat-destinatario.
Nombre:	EliminarDestinatario()
Descripción:	Esta responsabilidad permite eliminar un destinatario de la base de datos utilizando la clase dat-destinatario.
Nombre:	ExisteDestinatario()
Descripción:	Esta responsabilidad permite saber si ya existe un destinatario en la base de datos con el objetivo de no duplicar destinatarios de manera que el aviso que se está insertando le llegue una sola vez al destinatario.
Nombre:	ModificarDestinatario()
Descripción:	Esta responsabilidad permite modificar un destinatario de la base de datos utilizando la clase dat-destinatario.

2.5.6 Descripciones de las clases de diseño.

Tabla 20 Descripción de la clase ContenidoBModel

Nombre: ContenidoBModel	
Tipo de clase:	
Para cada responsabilidad:	
Nombre:	AgregarContenido()
Descripción:	Esta responsabilidad permite adicionar un contenido a la base de datos utilizando la clase dat-contenido.
Nombre:	EliminarContenido()
Descripción:	Esta responsabilidad permite eliminar un destinatario de la base de datos utilizando la clase dat-destinatario.
Nombre:	ModificarContenido()
Descripción:	Esta responsabilidad permite modificar un contenido de la base de datos utilizando la clase dat-contenido.

2.5.7 Descripciones de las clases de diseño.

Tabla 21 Descripción de la clase GestEventosController

Nombre: GestEventosController	
Tipo de clase:	
Para cada responsabilidad:	
Nombre:	AgregarEventos()
Descripción:	Esta responsabilidad permite agregar un evento al sistema.
Nombre:	ModificarEventos()
Descripción:	Esta responsabilidad permite modificar un evento del sistema.
Nombre:	EliminarEventos()
Descripción:	Esta responsabilidad permite eliminar un evento del sistema.
Nombre:	listarEventos()
Descripción:	Esta responsabilidad permite listar los eventos en la interfaz de eventos.

2.6 Patrones utilizados

El empleo de los patrones GRASP tiene como objetivo la descripción de los principios fundamentales de diseño de objetos para la asignación general de responsabilidades. Para la asignación general de responsabilidades del componente Alertas y Avisos se utilizaron los que se listan a continuación:

- Alta cohesión.
- Bajo acoplamiento.

Dentro del grupo de GOF se utilizaron el Fachada y el Singleton en el IoC Inversión de control (Inversion of Control) funcionando como intermediario para la comunicación entre componentes. Además se utiliza el patrón observador para monitorear los eventos del sistema y finalmente dentro de los patrones arquitectónicos se utiliza el Modelo Vista Controlador para separar en tres capas diferentes los datos de la aplicación, la interfaz de usuario, y la lógica de control

2.7 Diagrama de componentes.

Lo que distingue a un diagrama de componentes de otros tipos de diagramas es su contenido. Normalmente contienen componentes, interfaces y relaciones entre ellos. Y como todos los diagramas, también puede contener paquetes utilizados para agrupar elementos del modelo.

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes software, sean éstos componentes de código fuente, binarios o ejecutables. Desde el punto de vista del diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización, y las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo. Los elementos de modelado dentro de un diagrama de componentes serán componentes y paquetes. (23)

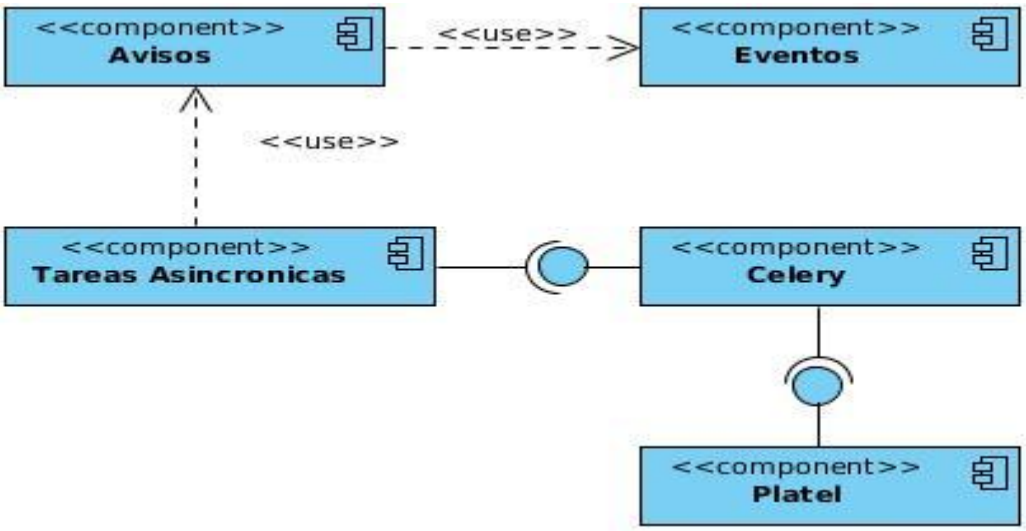


Figura 9. Diagrama de componentes Alertas y Avisos

El diagrama representado en la Figura está compuesto por el componente Avisos que se encarga de gestionar los avisos, utilizando el componente eventos que brinda los eventos que se disparan en el sistema en tiempo real, cuenta además con el componente tareas asíncronas que gestiona el monitoreo de los avisos y le brinda la información al componente Celery que se encarga de garantizar la sincronía entre el componente tareas asíncronas y Platel que es el servidor.

2.8 Modelo de datos

Un modelo de datos: Es un conjunto de conceptos que nos permiten describir los datos, las relaciones que existe entre ellos, la semántica y las restricciones de consistencia (24).

Los modelos de datos aportan la base conceptual para diseñar aplicaciones que hacen un uso intensivo de datos, así como la base formal para las herramientas y técnicas empleadas en el desarrollo y uso de sistemas de información. Con respecto al diseño de bases de datos, el modelado de datos puede ser descrito dados los requerimientos de información y proceso de una aplicación de uso intensivo de datos (por ejemplo, un sistema de información), construir una representación de la aplicación que capture las propiedades estáticas y dinámicas requeridas para dar soporte a los procesos deseados (por ejemplo, transacciones y consultas). Además de capturar las necesidades dadas en el momento de la etapa de diseño, la representación debe ser capaz de dar cabida a eventuales futuros requerimientos". (25)

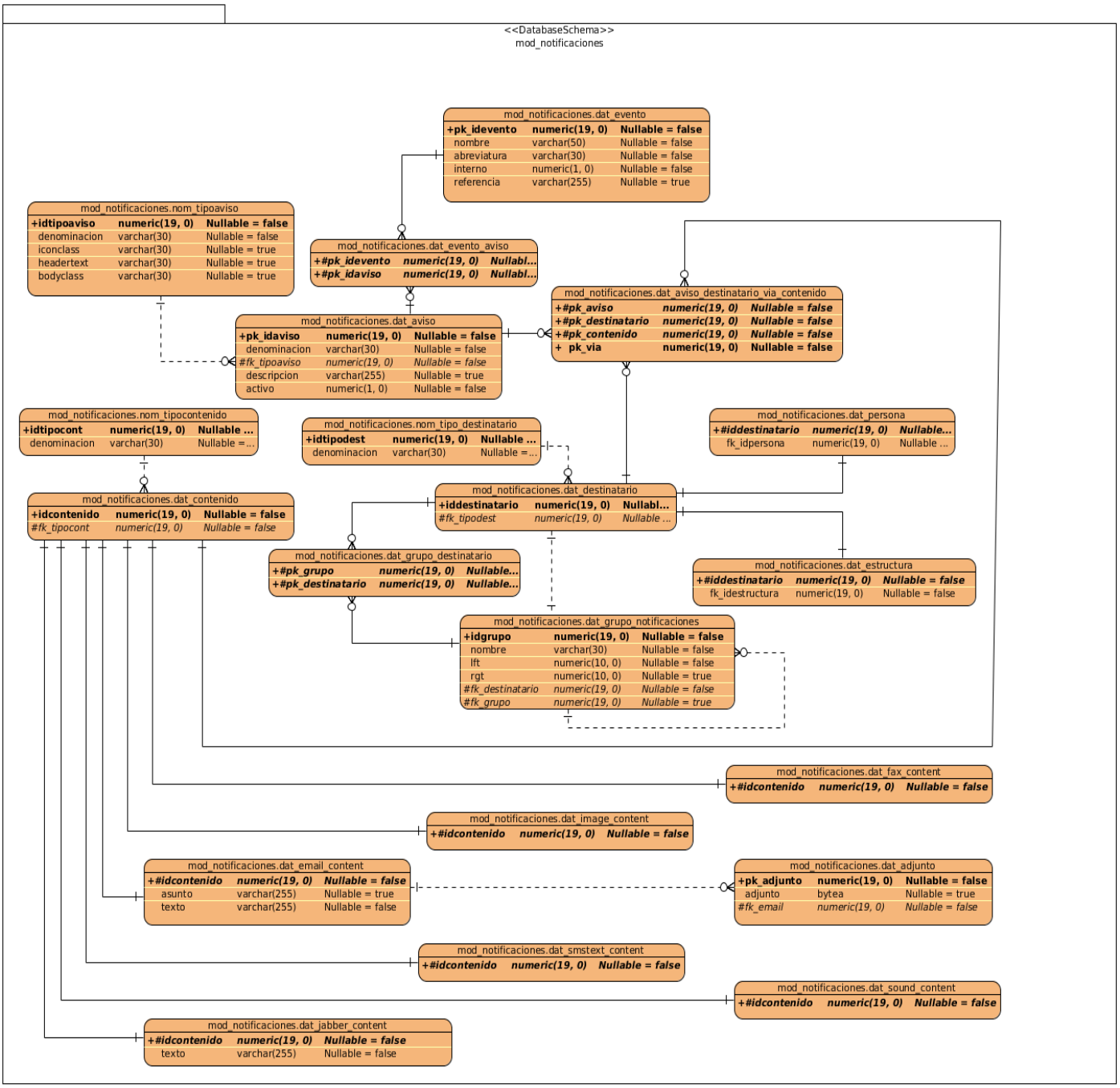


Figura 10. Diagrama de datos

Resumen de tablas de modelo de datos

Tabla 22 Descripción de tablas de modelo de datos

mod_notificaciones	
dat_evento	Almacena la información de los eventos tanto del sistema como externos.
nom_tipoaviso	Almacena los tipos de avisos que pueden existir
dat_evento_aviso	Almacena los eventos y los avisos de estos.
dat_aviso_destinatario_via_contenido	Almacena la referencia de los avisos con sus destinatarios, Vías y contenidos
dat_aviso	Almacena los avisos o notificaciones.
nom_tipocontenido	Almacena los tipos de contenido que se pueden enviar
dat_persona	Almacena los destinatarios que son personas de capital humano.
nom_tipo_destinatario	Almacena los tipos de destinatarios que pueden recibir notificaciones. Ej.: Personas, Entidades.
dat_destinatario	Almacena los destinatarios de los avisos.
dat_contenido	Almacena los contenidos de los avisos
dat_grupo_destinatario	Almacena los grupos con sus respectivos destinatarios
dat_estructura	Almacena los destinatarios que son estructuras.
dat_grupo_notificaciones	Almacena los grupos de destinatarios.
dat_fax_content	Almacena los contenidos de tipo fax.
dat_image_content	Almacena los contenidos de imagen.
dat_email_content	Almacena los contenidos de correo.
dat_adjunto	Almacena los adjuntos de email.
dat_smstext_content	Almacena los contenidos de tipo texto de sms.

dat_sound_content	Almacena los contenidos de tipo sonido.
dat_jabber_content	Almacena los contenidos de tipo jabber.

CONCLUSIONES PARCIALES

En este capítulo se determinaron los requisitos funcionales del sistema y se describieron los mismos además se explica el diseño realizado el cual tiene como principal objetivo modelar y describir los diagramas de clases del diseño, realizar el diagrama de componentes y el diagrama de base de datos correspondiente, de esta manera queda lista la propuesta del sistema y da paso a la realización posterior de la implementación.

Capítulo 3. Validación de clases y requisitos.

3.1 INTRODUCCIÓN

En el presente capítulo se realizará la validación de los requisitos, ya que de esa forma se muestra que los requisitos definen el sistema que realmente el usuario desea. Se utilizarán para la validación técnicas y métricas.

3.2 Técnicas para la validación de los requisitos.

Los requisitos una vez definidos necesitan ser validados. La validación de requisitos tiene como misión demostrar que la definición de los requisitos especifica realmente el sistema que el usuario necesita o el cliente desea. Es necesario asegurar que el análisis realizado y los resultados obtenidos de la etapa de definición de los requisitos son correctos. Pocas son las propuestas existentes que ofrecen técnicas para la realización de la validación y algunas de ellas consisten en revisar los modelos obtenidos en la definición de requisitos con el usuario para detectar errores o inconsistencias.

Reviciones: Está técnica consiste en la lectura y corrección de la completa documentación o modelado de la definición de requisitos. Con ello solamente se puede validar la correcta interpretación de la información transmitida. Más difícil es verificar consistencia de la documentación o información faltante.

Listas de chequeo: Son frecuentemente usadas en inspecciones o revisiones de artefactos generados en el proceso de producción de software; son listas de aspectos que deben ser completados o verificados.

Auditorías: La revisión de la documentación con esta técnica consiste en un chequeo de los resultados contra una checklist (Listas de Chequeo) predefinida o definida a comienzos del proceso, es decir sólo una muestra es revisada.

Prototipos: Algunas propuestas se basan en obtener de la definición de requisitos prototipos que, sin tener la totalidad de la funcionalidad del sistema, permitan al usuario hacerse una idea de la estructura de la interfaz del sistema con el usuario.

Un prototipo es una versión inicial de un sistema de software que se utiliza para demostrar los conceptos, probar las opciones de diseño y entender mejor el problema y su solución.

Un prototipo puede revelar errores u omisiones en los requerimientos propuestos, favorece la comunicación entre clientes y desarrolladores, da una primera visión del producto.

Esta técnica tiene el problema de que el usuario debe entender que lo que está viendo es un prototipo y no el sistema final.
La técnica utilizada fue la de prototipos para un mejor entendimiento del cliente y los analistas, de lo que se quería que hiciera el sistema.

3.3 Métricas para la validación de los requisitos.

Pasos para la validación de requisitos.

Esta actividad tiene como propósito validar que se ha descrito lo correcto y verificar que se ha hecho cabalmente.
La validación se realiza en tres pasos:

a) Revisión Técnica por el equipo de Analistas Principales. El objetivo de este paso es verificar que se hayan construido correctamente los artefactos correspondientes a la Ingeniería de Requisitos.

b) Revisión Funcional. Se realizará con el propósito de validar que las funcionalidades descritas satisfacen las expectativas de los interesados y que desde el punto de vista funcional se han descrito correctamente.

c) Taller de Aprobación. El objetivo de este taller es presentar las soluciones a las observaciones detectadas en los pasos anteriores y aprobar la Especificación de requisitos según los criterios para la evaluación y aceptación de los requisitos.

Si como resultado de este taller aún persisten observaciones y se decide aprobar los requisitos, se llegará a un acuerdo de cómo resolver cada una. Los acuerdos se registran en la sección observaciones de la Especificación de requisitos y se procede a su aprobación. Una vez aprobados los requisitos estos no pueden ser modificados, pero el equipo de desarrollo está obligado a resolver en el sistema las observaciones registradas y a actualizar la Especificación de requisitos.

Criterios para la evaluación y aceptación de los requisitos mediante métricas.

Durante las validaciones hay que registrar las observaciones realizadas y clasificarlas según su tipo en: de negocio, formato, ortografía y redacción, consistencia y otros.

De negocio: Son las observaciones realizadas porque el requisito omite o no refleja correctamente algún aspecto del negocio, por ejemplo: atributos de un concepto erróneos, omisión de validaciones y restricciones de negocio, omisión de un requisito.
Formato: Son las observaciones realizadas porque se han omitido secciones requeridas de la plantilla.
Ortografía y redacción: Son las observaciones realizadas por concepto de errores ortográficos o de redacción.
Consistencia: Son las observaciones realizadas por contradicciones existentes entre los requisitos.
Otros: Son las observaciones que no se ajustan a ninguna de las categorías anteriores.
Para aceptar una Especificación de requisitos debe cumplirse que:
La correctitud sea menor que 0,10.
La completitud sea mayor que 0,90.
La consistencia se menor que 0,20.

Observaciones de negocio tomadas en la revisión técnica formal de los requisitos del sistema:

- Omisión de las validaciones de los requisitos
- Omisión de flujos alternos

Correctitud

La métrica se propone determinar si la Especificación de requisitos contiene todos los requisitos necesarios para satisfacer las necesidades del negocio y los interesados.
X = D/T
X – Correctitud.
D – Total de observaciones de negocio realizadas en la validación.
T – Total de requisitos revisados.

Correctitud
X = D/T
X = 2/21
X = 0.09

Completitud

La métrica pretende determinar si la Especificación de requisitos es completa.

X = 1- O/S

X – Completitud

O – Total de observaciones de formato.

S – Total de secciones del documento.

Completitud
X = 1-(O/S)
X = 1-(0/6)
X = 1

Consistencia

La métrica se propone determinar si la Especificación de requisitos es consistente.

X = C/T

X – Consistencia

C – Total de observaciones de consistencia

T – Total de requisitos revisados.

Consistencia
X = C/T
X = 0/21
X = 0

Al aplicar las métricas a los requisitos funcionales definidos se cumple que las métricas correctitud, completitud y consistencia cumplen las restricciones para la aceptación de la especificación de los requisitos.

3.4 Validación de las clases del diseño

Las métricas de clases en sentido general, permiten averiguar cuán bien están definidas las clases y el sistema, lo cual tiene un impacto directo en el mantenimiento del mismo, tanto por la comprensión de lo desarrollado como por la dificultad de modificarlo con éxito.

Las métricas empleadas Tamaño operacional de la clase y Relaciones entre clases están diseñadas para evaluar los siguientes atributos de calidad:

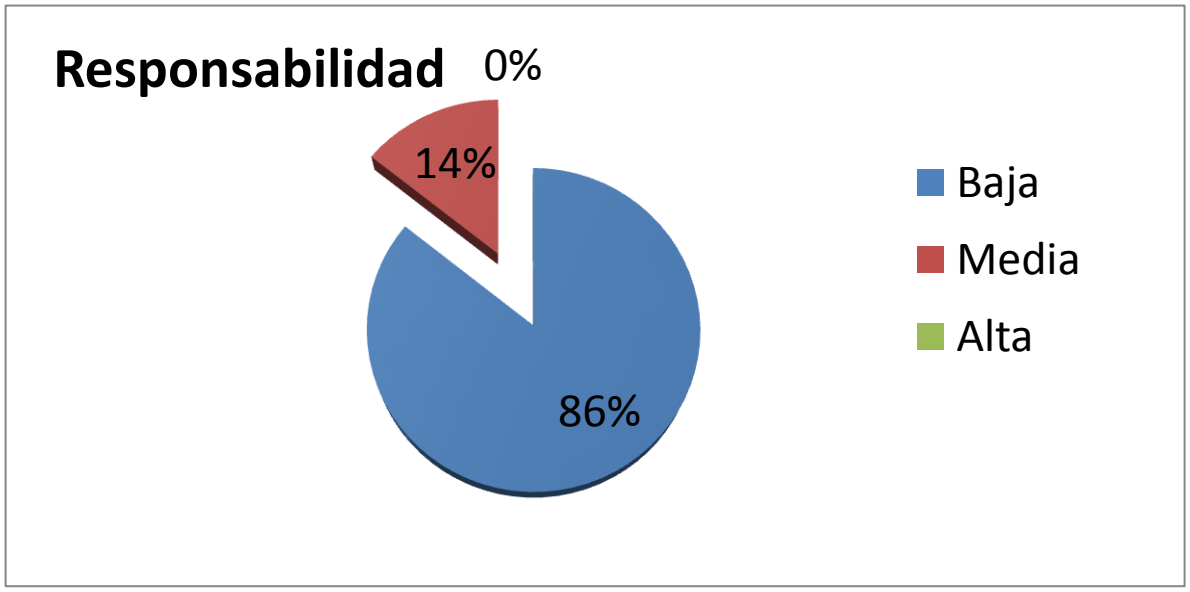
- ***Responsabilidad.*** Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- ***Complejidad de implementación.*** Consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- ***Reutilización.*** Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- ***Acoplamiento.*** Consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.
- ***Complejidad del mantenimiento.*** Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
- ***Cantidad de pruebas.*** Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (componente, modulo, clase, conjunto de clases, etc.) diseñado.

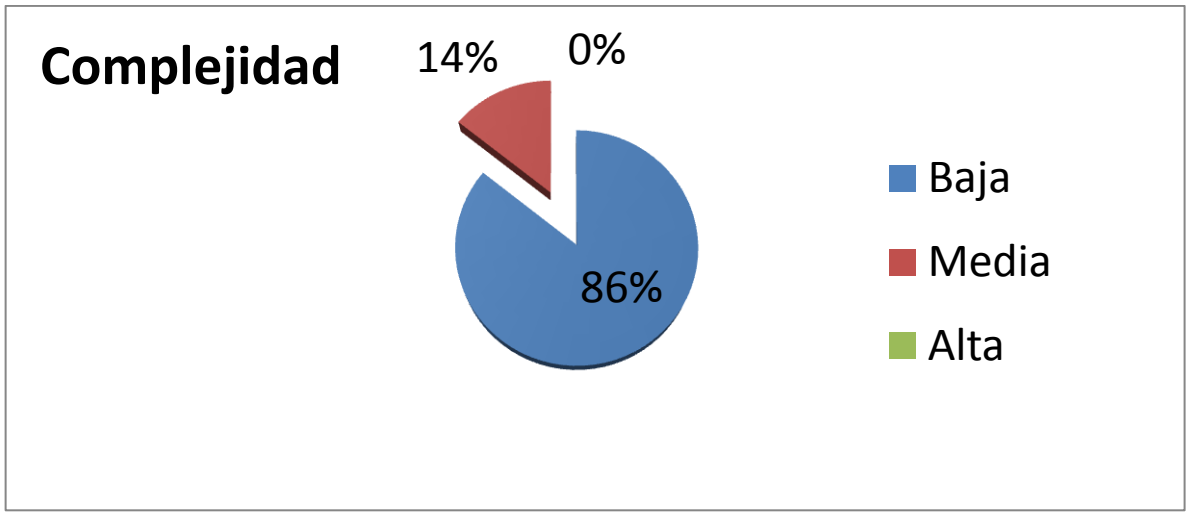
Tamaño operacional de la clase

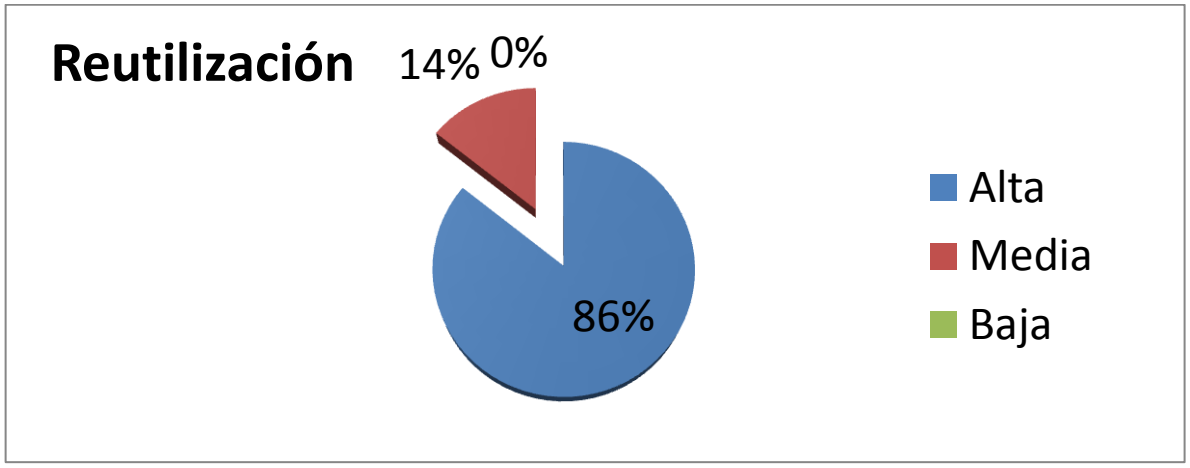
Tamaño operacional de clase (TOC): Está dado por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad:

Atributo de calidad	Modo en que lo afecta
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución del grado de reutilización de la clase.

Clases	Cantidad de Métodos	Responsabilidad	Complejidad	Reutilización
ContenidoBModel	4	Baja	Baja	Alta
GestAvisosController	10	Media	Media	Media
AvisoBModel	4	Baja	Baja	Alta
EventoBModel	4	Baja	Baja	Alta
GrupoBModel	6	Baja	Baja	Alta
DestinatarioBModel	4	Baja	Baja	Alta
GestEventosController	4	Baja	Baja	Alta







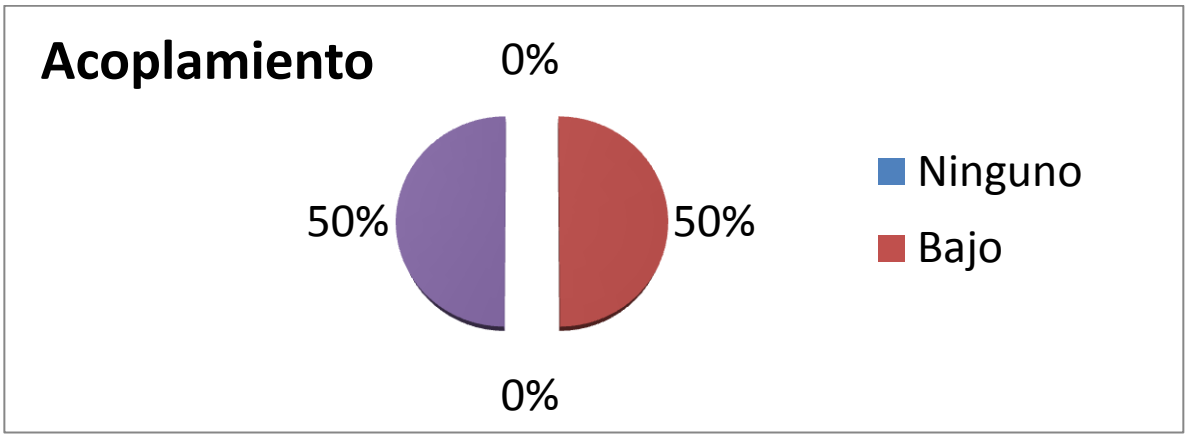
Una vez obtenidos los resultados de la evaluación del instrumento de medición de la métrica TOC, se puede concluir que el diseño propuesto tiene una calidad aceptable teniendo en cuenta que el 86 % de las clases tienen la responsabilidad y la complejidad baja y la reutilización es alta lo que garantiza que cualquier modificación o reutilización en el sistema sea eficaz.

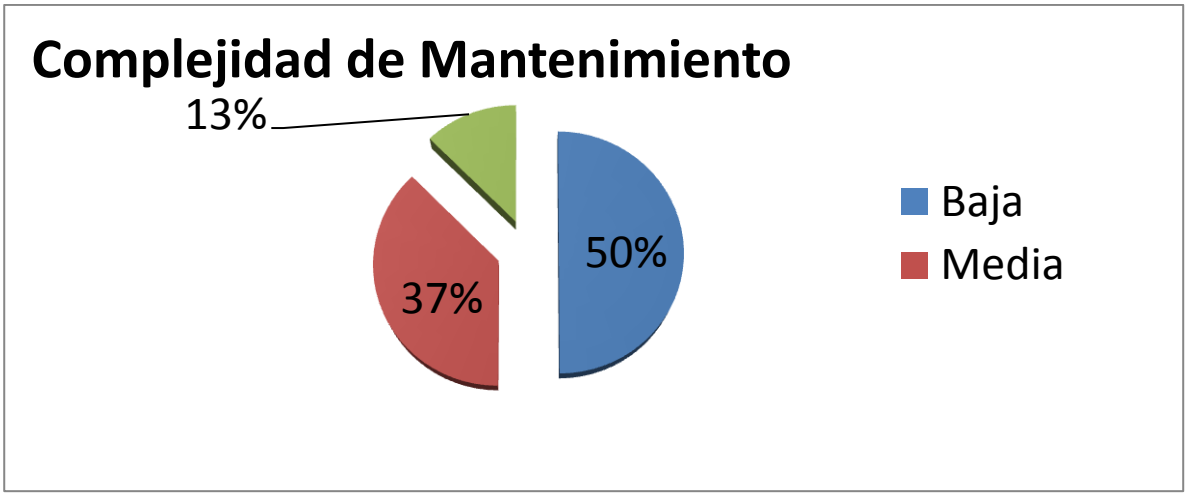
Relaciones entre clases

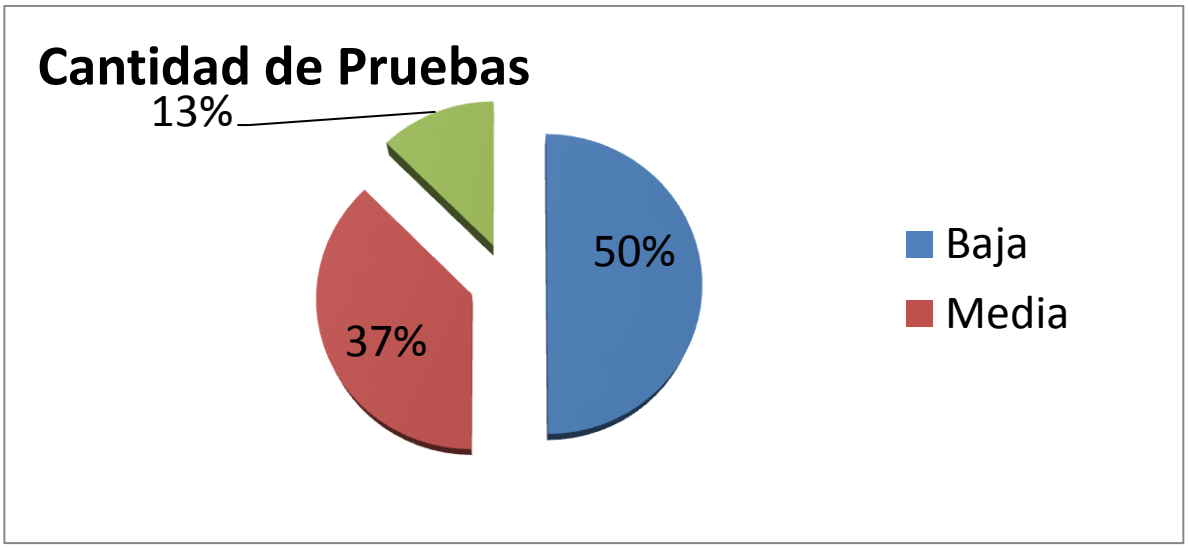
Relaciones entre clases (RC): Está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad:

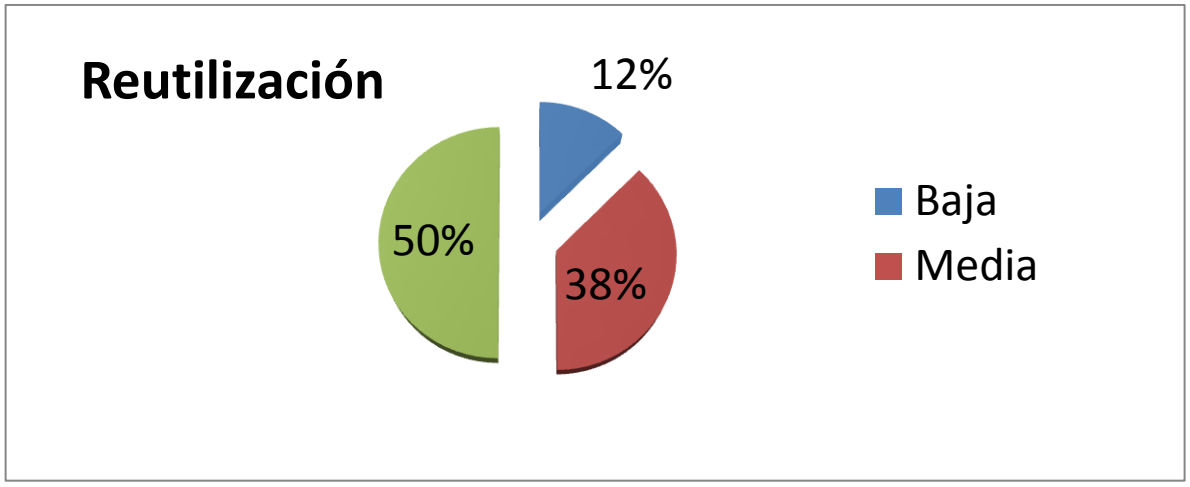
Atributo de calidad	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del Acoplamiento de la clase.
Complejidad de mantenimiento	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
Reutilización	Un aumento del RC implica una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Clases	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad Mant.	Reutilización	Cantidad de Pruebas
ContenidoBMoel	1	Bajo	Baja	Alta	Baja
GestAvisosController	5	Alto	Alta	Baja	Alta
AvisoBModel	1	Bajo	Baja	Alta	Baja
EventoBModel	1	Bajo	Baja	Alta	Baja
GrupoBModel	3	Alto	Media	Media	Media
DestinatarioBModel	3	Alto	Media	Media	Media
ContactoBModel	3	Alto	Media	Media	Media
GestEventosController	1	Bajo	Baja	Alta	Baja


Acoplamiento
0%
50%
50%
0%
Ninguno
Bajo


Complejidad de Mantenimiento
13%
50%
37%
Baja
Media





Una vez obtenidos los resultados de la evaluación del instrumento de medición de la métrica RC, se puede concluir que el diseño propuesto tiene una calidad aceptable teniendo en cuenta que el 50% de las clases empleadas posee menos de 3 dependencias de otras clases lo que conlleva a evaluaciones positivas de los atributos de calidad involucrados (acoplamiento, complejidad de mantenimiento, cantidad de pruebas y reutilización). Favoreciendo de esta manera la reutilización de las clases así como la modificación e implantación del diseño

Al aplicar las métricas a las clases del diseño definidas se cumple que las métricas de Tamaño operacional de la clase y Relaciones entre clases cumplen las restricciones para la aceptación de las clases.

3.5 CONCLUSIONES.

La validación de los requerimientos es una actividad muy importante, pues un levantamiento de requerimientos con errores que no se detecten a tiempo, además de conducir a resultados inesperados, provoca costos excesivos y gran pérdida de tiempo.

Mediante los resultados arrojados por las métricas y técnicas utilizadas para la validación de requisitos, así como la validación de las clases del diseño se pudo verificar que los requisitos y las clases tienen la calidad requerida para su aprobación.

CONCLUSIONES GENERALES:

Después de realizada la investigación se arribaron a las siguientes conclusiones:

En este trabajo se demostró la necesidad de realizar el análisis y diseño de un componente de Alertas y Avisos que fuese capaz de adaptarse a las necesidades del Marco de Trabajo Sauxe. La firma de los requisitos, así como las métricas empleadas validaron que se realizó una correcta gestión de los mismos así como la aplicación de las métricas orientadas a clases, TOC y RC para la validación del diseño, demostraron que se realizó un buen diseño de clases. Finalmente con el diseño propuesto se puede proceder a la implementación del componente Alertas y Avisos.

RECOMENDACIONES

- Utilizar el presente trabajo como bibliografía para posibles investigaciones referentes al tema desarrollado en el mismo.
- Se recomienda que el resultado propuesto sirva como punto de partida para la posterior implementación del componente Alertas y Avisos.
- Se exhorta a que constituya el elemento fundamental de valoración en cuanto a futuros cambios.

TRABAJOS CITADOS

1. *DIAZ, FRANCISCO PAUL HERRERA. http://paulhdcpp.wordpress.com/2010/04/12/deberes/. [En línea] UNIVERSIDAD POLITECNICA SALESIANA, 10 de Abril de 2010.*

2. sevilla, universidad de. Introducción a las aplicaciones web. 2008.

3. Pedro Boda, K.N., Javier Martinez, Benjamín Gonzales. Zend Framework Manual en Español. 2009.

4. Kaplan-Moss, Adrian Holovaty y Jacob. El libro de Django . 2008.

5. RO, MA S S IMO D I P I E R. manual de web2py. 2007.

6. Hibbs, Curts. http://oreilly.com/ruby/archive/rails.html. [En línea] 2009.

7. Leopoldo, Carlos. ZendFramework, introducción . http://techtastico.com/post/zend-framework-una-introduccion/. [En línea] 2007.

8. Gomez, Oiner. Plantilla Registro de la Propiedad intelectual(Sauxe). 2008.

9. Doctrine, ServerGove. http://www.doctrine-project.org/. [En línea] 2010.

10. Corso, Giancarlo. lo bueno, lo malo y lo feo. . 22 de Octubre de 2008.

11. Ivar Jacobson, Grady Booch, James Rumbaugh. El Proceso Unificado de Desarrollo de Softaware. 2005.

12. Gestión de proyectos ágil: conceptos básicos. Palacio, Juan. 2006.

13. Amaro Calderón, Sarah Dámaris, Valverde Rebaza y Jorge Carlos. Metodologías Ágiles. Trujillo – Perú : s.n., 2007.

14. Pressman, Roger. Ingenieria del Software un enfoque práctico. La Habana: Felix Varela. 2005.

15. desarrollo, Proyecto de. Modelo de Desarrollo orientado a componentes del proyecto ERP -CUBA. 2008.

16. Rumbaugh, James, Jacobson, Ivar y Booch, Grady. El Lenguaje Modificado de Modelado. 2000.

17. *Jacobson, Ivar, Boch, Grady y Rumbaugh, James. El Proceso Unificado de Desarrollo de Software. 2005.*

18. Loureiro, Tania Teresa. Análisi y diseño de la solución informática para el subsistema de Caja, del sistema de gestión empresarial Cedrux. 2009.

19. IBM. http://www.ibm.com/ 2010.

20. GSINNOVA, Grupo de Soluciones. http://www.rational.com.ar/index.html 2011.

21. Intelego. http://www.intelegosoft.com/esp/ea/index.asp 2005.

22. visual-paradigm. http://www.visual-paradigm.com/ visual paradigm. [En línea] 2010.

23. Daniele, Ing.Marcela. El arte de modelar . 2007.

24. Modelo de datos. http://aurea.es/wp-content/uploads/modelodedatos.pdf. [En línea]

25. Ortiz, Antonio Moreno. ISSN: 1139-8736 2000.

BIBLIOGRAFíA

Baryolo, Oiner. XVI Fórum de Ciencia y Técnica. 2009.

By Martin Owen and Jog Raj, Popkin. Software BPMN and Business Process Management. 2005.

Ruiz, Francisco. Facultad de Ciencias Económicas. http://www.economicasunp.edu.ar/02-EGrado/materias/trelew/analisis_sistemas%20II/info/Tecnol. [En línea] 14 de Enero de 2009.

Smith, L. Introduction to the Doctrine Object Relational Mapper. 04/2009.

Software, Corporatio. Rational Rational Unified Extended Help . 2003.

Amaro Calderón, Sarah Dámaris, Valverde Rebaza y Jorge Carlos. Metodologías Ágiles 2007.

http://www.gestionyadministracion.com/empresas/concepto-de-gestion.html. http://www.gestionyadministracion.com/empresas/concepto-de-gestion.html. [En línea] 2008.

JIMÉNEZ, JUAN JOSÉ. GESTIÓN EMPRESARIAL. PROYECTO CONFLICTOS INTERCULTURALES: UNA RESPUESTA DEMOCRÁTICA Y PARTICIPATIVA REGIONAL DESDE BOLIVIA, ECUADOR Y PERÚ. La Paz : De encuentro, 2008.

CEDRUX. Documento Especificación de la Arquitectura de Sistema e Integración ERP Cuba. 2009.

IEEE. ESPECIFICACIONES DE LOS REQUISITOS DEL SOFTWARE. 1998.