UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

Componente gestión de notificaciones del marco de trabajo sauxe basado en reglas de negocio.

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

Autor:

Osbel Rodríguez Corrales

Tutores:

MSc. Orlando Arnaldo Valenzuela Aguilera

Ing. Yuniel Cedeño Mendoza

La Habana, Junio de 2014

"Año 56 de la Revolución"


“Solo el conocimiento que llega desde dentro es el verdadero conocimiento.”
Sócrates

DECLARACIÓN DE AUTORÍA

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

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

_____

Firma del Autor

A mi mamá y mi papá por estar siempre presentes cuando los he necesitado, por darme la formación que hoy tengo, por todo el amor y cariño que siempre me han dado, y por el sacrificio incondicional que han hecho para poder sacarme adelante y encaminarme en la vida, a mi mama por todos los consejos que me dio a lo largo de mis estudios. Los quiero con todas las fuerzas de mi corazón mami y papi.

A mi hermano Osmel por toda la ayuda que me ha dado y por estar siempre ahí cuando lo he necesitado, lo quiero con la vida.

A mi tío José, a mis abuelos Mirtha y Antonio, a Julián y al resto de mi familia que me han dado toda mi formación como persona, por el amor y educación, y por estar siempre conmigo en las buenas y malas dándome mucho apoyo y consejos para salir adelante.

A mi tía Oneida y mi primo Alien por toda la ayuda y atención que me dieron en mis cinco años.

A mi tía Damaris y Acuña por toda la ayuda que me dieron.

A mi novia Bárbara Salinas y su familia en general por todo el amor y cariño que me dieron y por las atenciones que han tenido conmigo, por estar presentes cuando los he necesitado. Los quiero muchísimo.

A Pedro Nogales que fue un guía y un ejemplo a seguir, que me dio mucha ayuda y dedicación en los años últimos años de la carrera, principalmente en la realización de la tesis.

A Yoansy López por toda la ayuda que me dio a mi como a mi mama, y por el apoyo y ayuda brindado.

A mis tutores por siempre estar ahí cuando los he necesitado y por el apoyo en la confección en este trabajo.

A Yoriangel Rivero por transmitirme el conocimiento y explicarme dudas acerca de los lenguajes y herramientas de programación.

A Zoila Guerra y Roexcy Vega por toda la preocupación que han tenido conmigo y con mis padres, y la ayuda que nos han dado para solucionar cualquier problema en mi estancia en la Universidad.

A mis compañeros de estudios e infancia, que han venido conmigo desde muy temprana edad Karel Tase y su hermano Alex, estudiando siempre juntos para poder salir con buenos resultados en los exámenes y por toda la dedicación que me han brindado en cada uno de los años.

A todos mis compañeros de estudios que han estado de una forma u otra presentes y con quienes he convivido todos estos años y siempre han estado ahí de forma positiva.

A todos los profesores que me han impartido clases a lo largo de la carrera y que han plasmado su conocimiento tributando a mi formación como profesional y como persona.

A todas las personas que de una forma u otra hicieron posible que llegara a ser lo que hoy soy, muchas gracias a todos.

RESUMEN

En la actualidad los sistemas web desarrollados sobre el marco de trabajo Sauxe, debido a la arquitectura cliente servidor que poseen, no tienen forma de conocer la ocurrencia de algún evento en el servidor en el mismo instante en que este sucede y mostrarlo en una interfaz gráfica a través de dicho marco sin tener que intervenir en el correo o el jabber, además el componente de notificaciones no cuenta con un mecanismo en cuestión de reglas de negocio para restringir la ocurrencia de estos eventos.

En el presente trabajo se realiza la propuesta del componente de gestión de notificaciones del marco de trabajo sauxe basado en reglas de negocio, el cual permite a los usuarios y a los componentes de las aplicaciones desarrolladas sobre Sauxe, establecer una comunicación en tiempo real y el desarrollo de soluciones basadas en reglas de negocio.

En el desarrollo del trabajo se detalla cómo funciona el componente de gestión de notificaciones mostrando todas sus alertas en la página de inicio del marco de trabajo, además se definen las tecnologías y lenguajes utilizados para su implementación, así como la reglas de negocio necesaria para asociar las notificaciones mediante las acciones.

El desarrollo de las tecnologías de la información ha conllevado a un crecimiento acelerado de la competencia entre las empresas por el apoderamiento del mercado mundial. Las empresas necesitan constantemente mejorar sus procesos, pero frecuentemente están frenadas por aplicaciones y sistemas que no están preparados para explotar nuevas oportunidades y adaptarse a los cambios de forma ágil. La Gestión de Procesos de Negocio (por siglas en inglés BPM), es un conjunto de herramientas, tecnologías, técnicas, métodos y disciplinas de gestión para la identificación, modelación, análisis, ejecución, control y mejora de los procesos de negocio, permitiendo a las organizaciones la flexibilidad y agilidad necesarias para adaptarse a los rápidos y continuos movimientos del mercado (Club-BPM, 2009).

Las experiencias de muchas organizaciones que han implantado BPM reportan grandes beneficios como: mejorar la atención y servicio al cliente, incrementar el número de actividades ejecutadas en paralelo, disminución "drástica" del tiempo de transferencia de trabajo, información y documentos entre actividades, asegurar la continua participación y colaboración de todo el personal en el proceso, disponer de mecanismos para una mejor gestión y optimización de procesos; con ahorros en costos y reducciones importantes en tiempos de servicios a sus clientes, dándose cuenta que BPM junto con sus tecnologías se hacen imprescindibles para convertir los retos en una realidad (Club-BPM, 2009).

En el ámbito de las aplicaciones existen 3 elementos que son: procesos, eventos y reglas, donde solo estaremos analizando las reglas del negocio que no es más que la declaración de políticas y restricciones de negocio de una organización, forman parte del código fuente del programa y son implementadas por un desarrollador en cualquier lenguaje de programación y generalmente se pueden escribir con sentencias de la forma "if-then". Las reglas de negocio pueden cambiar con frecuencia debido a las modificaciones en la gestión de un proceso de negocio o a cambios legales por mencionar algunos. La aplicación de estas sugieren ventajas en cuanto al aumento de la modularidad, consistencia, simplicidad y auto-descriptiva entendible. Esto ha provocado, que en los últimos tiempos la tendencia en el desarrollo de soluciones se enfoca cada vez más a la implementación de sistemas basados en reglas de negocio, a estos se les conoce por sus siglas en inglés BRMS (Club-BPM, 2009).

Los BRMS son herramientas informáticas especializadas en BPM, que logran independizar las reglas de negocio de una institución de los procesos de negocio, que es el conjunto de todas las tareas y actividades coordinadas formalmente, dirigidas tanto por personas como por equipos, que lleva a conseguir un objetivo organizativo específico que se desarrollan en las mismas (GARIMELLA y otros, 2012), manteniendo una baja dependencia entre ambos. Permiten además realizar cambios en las reglas de negocio de estas instituciones sin modificar los procesos que se llevan a cabo en la misma (Hevia, 2013). Un sistema BRMS permite que los analistas del negocio puedan ver y entender las reglas sin tener que depender del departamento de informática para realizar las modificaciones (Chisholm, 2013).

En la Universidad de las Ciencias Informáticas (UCI) de Cuba, específicamente en el Centro de Informatización de Entidades (CEIGE) se desarrolla un marco de trabajo denominado “Sauxe”, cuyo objetivo es el desarrollo de aplicaciones web de gestión, lo cual sugiere la informatización de los procesos de negocio en las entidades cubanas. Este a su vez tiene implementado un componente de gestión de notificaciones, para garantizar que las aplicaciones web desarrollados en él, puedan notificar todos los eventos ocurridos tanto a usuarios, como a componentes concernientes a sus trabajos específicos, en el mismo instante en que ocurran en el lado del servidor, contribuyendo de esta forma a optimizar el tiempo y agilizar la toma de decisiones. Este componente brinda varias vías para enviar las notificaciones tales como: correo electrónico y jabber, y posee un propio buzón de mensajería siempre que se esté conectado a la aplicación. A pesar de ello, hoy existe la necesidad de establecer un mecanismo que en cuestiones de notificaciones restrinja la ocurrencia de eventos basándose en la modificación de un proceso de negocio, de manera que contribuya a brindar mayor información al usuario y al desarrollo de soluciones basadas en reglas de negocio.

Se define como **problema a resolver:** ¿Cómo aumentar los canales de notificación del marco de trabajo Sauxe para incrementar el nivel de acceso a la información al usuario?

Por lo cual se toma como **objeto de estudio**, soluciones para el manejo de notificaciones en software de gestión, enmarcado en el **campo de acción**, soluciones de gestión de notificaciones basadas en reglas de negocio. Quedando definido de esta manera el **objetivo general**, desarrollar un componente de gestión de notificaciones basado en reglas de negocio para incrementar el nivel de acceso a la información al usuario.

A partir del objetivo general se derivan los siguientes **Objetivos Específicos:**

- Definir la fundamentación teórica de los elementos más significativos en cuanto a la gestión de notificaciones.
- Realizar el análisis y diseño de las nuevas funcionalidades del componente de gestión de notificaciones del marco de trabajo Sauxe para incrementar el nivel de acceso a la información al usuario.
- Implementar un componente de gestión de notificaciones basado en reglas de negocio para incrementar el nivel de acceso a la información al usuario.
- Validar la solución propuesta mediante casos de prueba.

Para dar cumplimiento a los objetivos específicos se plantean las siguientes **tareas de investigación** a realizar:

- Valoración de los sistemas existentes relacionados con la gestión de notificaciones obteniendo funcionalidades idóneas para el desarrollo del componente propuesto.

- Análisis de los componentes del marco de trabajo Sauxe para una asimilación con la estructura y funcionamiento de los mismos.

- Desarrollo de artefactos que propone el modelo de desarrollo orientado a componentes de CEIGE enfocado al marco de trabajo Sauxe.

- Diseño de las interfaces de usuario correspondientes al desarrollo propuesto.

- Desarrollar las funcionalidades del componente teniendo en cuenta las buenas prácticas de programación.

- Validación de las funcionalidades de la aplicación a través de pruebas unitarias y aplicación en un entorno real.

Para el desarrollo de las tareas de investigación se utilizan Métodos de Científicos en la búsqueda y procesamiento de la información propuestos en (Roberto y otros, 2006). Los mismos se dividen en teóricos y empíricos. Los **Métodos Teóricos** son factibles en el estudio de las características poco observables del objeto de investigación. Dentro de este grupo se utilizan:

- El método **Histórico-Lógico**, posibilitó estudiar de forma analítica la trayectoria histórica real de los fenómenos, su evolución y desarrollo. El método permitió realizar la primera parte de la investigación haciendo un análisis bibliográfico de otros sistemas que usan componentes de notificación, el razonamiento basado en reglas de negocio y técnicas de programación más utilizadas en el desarrollo de estos tipos de componentes, dando paso a la exploración de trabajos realizados en el ámbito de notificar eventos en un sistema informático y de soluciones previas existentes a problemas similares al actual. Se utilizó para determinar a través de la evaluación de la bibliografía conceptos de esta temática, que permiten conocer el estado de la evolución actual del fenómeno e identificar posibles mejoras y alternativas de solución.

- El **Analítico-Sintético**, a partir de fuentes bibliográficas seguras se utilizó para el estudio de conceptos y técnicas con soluciones previamente desarrolladas. Permitiendo además descomponer el problema de investigación en elementos por separado y profundizar en el estudio de cada uno de ellos, para luego sintetizarlos en la solución propuesta.

Los **métodos Empíricos** tienen gran importancia ya que permitieron efectuar el análisis preliminar de la información, así como verificar y comprobar las concepciones teóricas. Dentro de este grupo se utiliza:

- El **método de Entrevista**: Se aplicó para obtener información sobre el manejo de las notificaciones basadas en reglas de negocio mediante las entrevistas efectuadas a los especialistas del centro CEIGE. Además para obtener los requisitos con que debe cumplir el componente.

Se define la siguiente **estructura del documento**, quedando conformado en tres capítulos:

Capítulo 1: Fundamentación Teórica. Se realiza un estudio del estado del arte sobre las reglas de negocio y algunas aplicaciones web que notifican sus alertas mediante estas. Se describen tanto las tecnologías, metodologías y herramientas definidas por el modelo de desarrollo del centro CEIGE para el desarrollo de sistemas.

Capítulo 2: Propuesta de solución. Se modela el negocio, se definen los requisitos funcionales y no funcionales con que contará el componente para la gestión de notificaciones basado en reglas de negocio en el marco de trabajo Sauxe. Además del análisis y diseño donde se define la arquitectura de datos y del sistema para generar los artefactos definidos por el modelo de desarrollo, y por último se describen los patrones de diseño y arquitectónicos definidos para el desarrollo de la solución.

Capítulo 3: Implementación y Pruebas. Se especifican aspectos de interés para el proceso de implementación tales, como los estándares codificación para el código fuente del sistema, el diagrama de despliegue donde se identifican los nodos necesarios para el despliegue de la solución. Se especifican además los resultados de la validación del componente mediante pruebas de software realizadas y finalmente se valora que beneficios trae el resultado obtenido en dichas pruebas.

FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se hace referencia a la fundamentación teórica del trabajo donde se realiza un estudio del estado del arte sobre las reglas de negocio, para resolver el problema que da razón a este trabajo. Además del estudio de algunos Sistemas de Gestión de Notificaciones basados en reglas de negocio. Se detallan algunas características fundamentales del modelo de desarrollo a utilizar y se exponen los respectivos artefactos generados por dicho modelo durante las primeras fases de desarrollo del trabajo. Finalmente se caracterizan las técnicas y herramientas que define CEIGE.

1.2 Reglas de Negocio

No existe una definición única para el término regla de negocio (Business Rule).
Algunos autores las definen como: “una sentencia que define o restringe algún aspecto del negocio” (Hay y otros, 1997).
De manera informal, se pueden concebir las reglas de negocio como un conjunto de condiciones que expresan cómo deben llevarse a cabo los procesos del negocio, de forma tal que su resultado sea aceptable. Describen o representan restricciones sobre el comportamiento del negocio (Ross, 2005).
En términos generales, las reglas de negocio pueden describir instrucciones que: “emprendan una acción o un conjunto de acciones, si hay una condición o combinación de condiciones o si se produce un suceso o una combinación de sucesos”.
Representan un conjunto de prácticas de negocio estándar que se deben de aplicar en forma consistente dentro de las actividades del negocio. Son el componente más dinámico de cualquier aplicación. Por lo tanto, su identificación y externalización mejora la capacidad de adaptación de las organizaciones a cambios en la industria y competencia.
Las reglas de negocio deben ser expresadas declarativamente, no en forma procedural; deben indicar lo qué se quiere asegurar sin indicar cómo. Es posible declararlas en lenguajes formales, pero también deben ser expresadas en “lenguaje natural”, facilitando su compresión para todos los involucrados en el negocio (Muñoz, 2007).

1.3 Clasificaciones de las Reglas de Negocio.

Una clasificación de alto nivel de las reglas de negocio se puede dividir en tres grandes grupos mutuamente excluyentes, detallados en la Figura 2 (Ross, 2010).

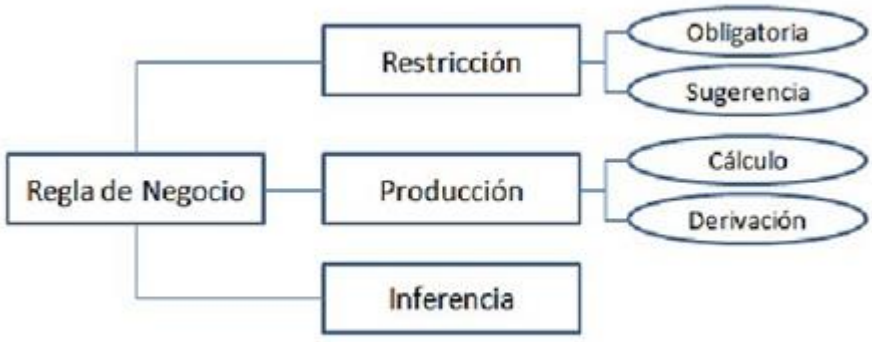


Figura 1. Clasificación de las Reglas de Negocio.

- Las reglas de negocio de tipo **Restricción** son una sentencia lógica que expresa una afirmación que debe ser válida en todo estado o transición entre estados, que pueden ser obligatorias o de sugerencia.

- Las reglas de negocio de tipo **Producción** calculan un valor o derivan un resultado, produciendo un cálculo en forma automática.

- Las reglas de negocio de tipo **Inferencia** son reglas que infieren nuevo conocimiento

- Regla de **cálculo**: calcula un valor por medio de una fórmula que contiene operadores estándar (suma, resta, división, multiplicación, promedio, etc.).

- Regla de **derivación**: el resultado de la evaluación es un valor lógico (verdadero, falso) y están basados en operaciones lógicas.

1.4 Sistemas basados en reglas de negocio.

1.4.1 ServiceDesk Plus

ServiceDesk Plus es un completo sistema ServiceDesk, basado en las "mejores prácticas" de Information Technology Infrastructure Library – (ITIL). Escalable, modular y con precios muy asequibles, ServiceDesk Plus es uno de los sistemas ServiceDesk más populares del mercado y cuenta con miles de usuarios en todo el mundo (incluyendo más de 650 empresas en España). A diferencia de otros productos de ServiceDesk, el mismo es fácil de implantar y su interfaz web es muy intuitivo. Un "wizard" le ayuda en cada paso de la implantación, lo que agiliza su puesta en marcha y reduce enormemente la dependencia de consultores externos y las notificaciones están basadas en reglas de negocio (Plus, 2009).

Este sistema cuenta con un mecanismo de "cascada", el cual busca a través de todas las reglas del negocio y ejecuta acciones para todas las reglas que se ajusten a la solicitud, en ocasiones las

solicitudes requieren de más de una regla del negocio. Además organiza las reglas brindando prioridad, acomodándolas en el orden en el que se quiera que se ejecuten. Si el mecanismo de "cascada" está deshabilitado, entonces ServiceDesk Plus no ejecutará la solicitud a través de otras reglas una vez que la misma sea cumplida. Envía las alertas vía email y/o SMS a técnicos en específico cada vez que una regla del negocio en particular es ejecutada. Estas reglas pueden ser creadas en base a criterios definidos personalizadamente. Si las condiciones de una regla son satisfactorias para la solicitud entrante, la regla se ejecuta y ServiceDesk Plus desempeña estas acciones pre definidas, sin embargo no muestra cómo realizar dichos procesos (Plus, 2009).

1.4.2 Sistema de Información Hospitalaria ALAS HIS.

El componente propone beneficios para todos los usuarios del Sistema de Información Hospitalaria ALAS (HIS) puesto que supone un aumento en la velocidad y precisión de la gestión de las alertas y notificaciones. Este sistema está compuesto por diferentes módulos donde la información está centralizada e interconectan las distintas áreas de una institución hospitalaria. Este sistema fue creado en el Centro de Informática Médica (CESIM) de la UCI con el objetivo de desarrollar soluciones informáticas para la salud. El sistema será desarrollado sobre herramientas multiplataforma lo que permite que pueda ser desplegado sobre entornos libres, garantizando un producto con mayor tiempo de vida, más reutilizable y eficiente. El componente constituye para Cuba una herramienta propia que responde a los intereses de sus organizaciones, y que está respaldada por un equipo de desarrollo cubano. Utiliza un Sistema de Gestión de Reglas de Negocio llamado Drools que consiste en un estándar para su motor de reglas de negocio. Drools es un framework para la construcción, mantenimiento, y la ejecución de las políticas de negocio en una organización, una aplicación o servicio.

Este sistema separa los datos y la funcionalidad de las aplicaciones, teniendo en cuenta que deben ser de fácil acceso, visibilidad, modificables y administrables, tanto para desarrolladores como para usuarios del negocio. La arquitectura que se emplea en este sistema es tres capas, lo que facilita a los programadores la labor de actualización y a los usuarios el uso del sistema (Villa, 2011). El sistema es implementado con Java y utilizando la librería Javamail para enviar las notificaciones vía correo; sus interfaces son creadas con Java Server Faces (JSF[1]) por sus siglas en inglés (Díaz y otros, 2013). Sin embargo Sauxe está desarrollado en PHP por lo que la manera en que las realizan la solución no es conveniente.

1.5 Modelo de desarrollo

Para el desarrollo de cualquier producto de software se realizan una serie de tareas entre la idea inicial y el producto final. Un modelo de desarrollo establece el orden en el que se harán las cosas

[1]**Java Server Faces (JSF):** es un framework para la construcción de interfaz de usuario que realiza la programación a través de componentes y basada en eventos.

en el proyecto, provee los requisitos de entrada y salida para cada una de las actividades (Ecured, 2011). Sommerville define el modelo de proceso de software como “Una representación simplificada de un proceso de software, representada desde una perspectiva específica. Por su naturaleza los modelos son simplificados, por lo tanto un modelo de procesos del software es una abstracción de un proceso real” (Sommerville, 2002).

1.5.1 Modelo de desarrollo propuesto

El departamento de Tecnología utiliza el Modelo de desarrollo de software definido por CEIGE, dicha definición incluye la especificación de las actividades de cada una de las fases del ciclo de vida de los proyectos del centro teniendo en cuenta los procesos de Capability Maturity Model Integration (CMMI) nivel 2 para la UCI. Para ello se toma la definición adoptada y publicada por el Software Engineering Institute (SEI) por sus siglas en inglés, como aval de la calidad de su proceso de desarrollo de software. Este modelo estandarizado establece las distintas fases por las que se debe transitar durante el desarrollo, y el conjunto de artefactos que se generan en cada una de ellas. Las fases definidas son: Inicio o Estudio preliminar y Desarrollo.

La solución que se desea implementar se enmarca en la fase de Desarrollo, en la cual se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, se identifican los requisitos, se elaboran la arquitectura y el diseño, se implementa, se hacen las pruebas internas y se libera el producto. El objetivo de esta fase es: Obtener un sistema que satisfaga las necesidades de los clientes y usuarios finales.

En la disciplina **modelado del negocio** se emplea el modelo de dominio, no se utiliza modelo de procesos debido a que en el sistema que se desea desarrollar no existen procesos de negocio definidos. En esta disciplina, para el modelo de dominio se genera el modelo conceptual, el cual describe los aspectos del dominio, identificando los principales elementos físicos o lógicos del negocio que ayuden a entender el problema y que generalmente se presentan como clases. El modelo conceptual explica cuáles son y cómo se relacionan los conceptos relevantes en la descripción del dominio de un problema, identificando atributos y asociaciones existentes entre ellos.

En la disciplina **Requisitos** se identifican los requisitos funcionales y no funcionales, se describen los mismos y se elaboran los prototipos de interfaces de usuario asociados a ellos.

Durante la disciplina **Análisis y diseño** se define el estilo y el patrón arquitectónico que determinan la estructura fundamental de la herramienta. Se genera el modelo de diseño, artefacto conformado por los diagramas de clases del diseño y secuencia con estereotipos web y el diagrama de componente. Por último se realiza el modelo de datos y se elaboran los diseños de casos de prueba.

En la disciplina **Implementación** se definen los estándares de codificación a utilizar y se implementan los requisitos funcionales identificados con sus correspondientes interfaces.
Por último, en la disciplina **Pruebas** se realizan pruebas de caja negra mediante la técnica de aceptación al cliente y las pruebas de caja blanca utilizando la técnica del camino básico, también se resuelven las no conformidades detectadas durante la ejecución de estas pruebas (CEIGE, 2013).
En general propone una solución sencilla y novedosa, que se centra en el desarrollo de componentes como base tecnológica, con una mayor calidad y en menor tiempo, para su posterior uso en la construcción de productos concretos; además propone dividir el trabajo y el equipo de desarrollo para lograr mayor especialización. Su buena aplicación proporcionará en gran medida la independencia tecnológica de los sistemas finales (Pérez, 2009).

1.5.2 Tecnologías propuestas

En CEIGE se realizó un estudio de las tecnologías que se utilizan durante todo el proceso de desarrollo, las cuales están referenciadas y brevemente explicadas a continuación:

1.5.3 Lenguajes de programación.

Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que se ponen a disposición del programador para que éste pueda comunicarse con los dispositivos hardware y software existentes (Escribano, 2002). A continuación se describen los lenguajes de programación a utilizados en el desarrollo del componente.

PHP v5.3.3

PHP (acrónimo de "PHP: HypertextPreprocessor") es un lenguaje de código abierto interpretado, de alto nivel, introducido en páginas HTML, que está completamente orientado al desarrollo de aplicaciones web interpretadas y ejecutadas del lado del servidor. Puede ser utilizado en casi todos los sistemas operativos existentes, tales como UNIX, Linux o Mac OS X y Windows, permitiendo así la migración a aplicaciones de un sistema a otro sin realizarse cambios en el código.
Su rapidez en la ejecución y los bajos requerimientos de consumo en los sistemas donde es desplegado lo hacen uno de los preferidos por los desarrolladores (Diaz, 2012).
Ventajas más significativas:

- Mejor soporte para la Programación Orientada a Objetos, que en versiones anteriores con PHP Data Objects, era extremadamente rudimentario.
- Mejoras de rendimiento.

- Mejor soporte a XML (XPath, DOM, etc.).
- Soporte integrado para SOAP.
- Iteradores de datos.
- Manejo de excepciones.

JavaScript v1.10.3

JavaScript es el lenguaje interpretado orientado a objetos desarrollado por Netscape que se utiliza en millones de páginas web y aplicaciones de servidor en todo el mundo. El mismo es un lenguaje de programación dinámico que soporta construcción de objetos basado en prototipos. La sintaxis básica es similar a Java y C++ con la intención de reducir el número de nuevos conceptos necesarios para aprender el lenguaje. No requiere de compilación ya que el lenguaje funciona del lado del cliente, y los navegadores son los encargados de interpretar estos códigos. JavaScript surgió con el objetivo de permitir a los autores de sitios web crear páginas que permitieran interactuar con los usuarios ya que HTML solo permitía crear páginas estáticas donde se podía mostrar textos con estilos, lo cual impedía interactuar con los usuarios (Eguíluz Pérez, 2009).

1.5.4 Lenguaje de modelado UML

Es una herramienta de diseño UML[2] libre y profesional, diseñado para contribuir al desarrollo de software. Soporta los principales estándares como UML, SysML, BPMN[3] y XML. Ofrece un completo conjunto de herramientas a los equipos de desarrollo de software, necesarios para la captura de requisitos, la planificación de software, la planificación de pruebas, el modelado de clases y el modelado de datos (Pressman, 2005).

1.5.5 Librerías y Marcos de Trabajo:

El proceso de implementación de la solución se efectuará utilizando el marco de trabajo Sauxe, desarrollado por el Departamento de Tecnología de CEIGE, el cual 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.

ExtJS v2.2

Es una librería Java Script de código abierto de alto rendimiento para la creación y desarrollo de aplicaciones web dinámicas. Provee interfaces gráficas de usuario que brindan experiencias

[2]**UML**: Lenguaje de Modelado Unificado.
[3]**BPMN:** Business Process Modeling Notation.

parecidas o iguales a las que se tienen con aplicaciones de escritorio. Permite la creación de aplicaciones complejas utilizando componentes predefinidos. Es extensible para la gran mayoría de los navegadores, evitando el tedioso problema de validar el código para cada uno de estos. Entre sus principales ventajas se encuentra el balance entre Cliente-Servidor, distribuyendo la carga de procesamiento en el último, y este al tener menor carga, maneja los clientes de manera más eficiente. La comunicación asíncrona permite el intercambio de información con el servidor sin necesidad de pedirle una acción al usuario, dando la libertad de cargar la información sin que este lo note (Frederick y otros, 2009).

Zend Framework 1.9.5

Zend Framework[4] es un marco de trabajo de código abierto para desarrollar aplicaciones y servicios web con PHP5. El mismo es una implementación que usa código 100% orientado a objetos. La estructura de los componentes de este marco de trabajo es única; cada componente está construido con una baja dependencia de otros componentes. Esta arquitectura débilmente acoplada permite a los desarrolladores utilizar los componentes por separado. A menudo se refiere a este tipo de diseño como "use-at-will" (uso a voluntad), aunque se pueden utilizar de forma individual; los componentes de la biblioteca estándar del mismo conforman un potente y extensible marco de trabajo de aplicaciones web al combinarse. Zend Framework ofrece un gran rendimiento y una robusta implementación Modelo Vista controlador (MVC), una abstracción de base de datos fácil de usar, y un componente que implementa la prestación de formularios HTML, validación y filtrado, para que los desarrolladores puedan consolidar todas las operaciones usando de una manera sencilla la interfaz orientada a objetos (Vaswani, 2010).

Doctrine 1.2.2

Es un mapeador de objetos relacional (ORM[5]) escrito en PHP que proporciona persistencia a sus objetos. Está por encima de la capa de abstracción a la base de datos, una de sus características es la posibilidad de escribir consultas a la base de datos a partir del tratamiento con objetos en PHP llamado Doctrine Query Language (DQL[6]), inspirado por Hibernates HQL[7]. Esto proporciona a los desarrolladores una poderosa alternativa a SQL[8] que mantiene la flexibilidad, sin necesidad de la duplicación de código innecesaria (Doctrine, 2007).

[4]**Framework**: en inglés “marco de trabajo”.
[5]**ORM** (Object-Relational Mapping): por sus siglas en español “Mapeo Objeto-Relacional”.
[6]**DQL** (Doctrine Query Language): por sus siglas en español “Lenguaje de Consulta de Doctrine”.
[7]**HQL** (Hibernate Query Language): por sus siglas en español “Lenguaje de Consulta de Hibernate”.
[8]**SQL:** *structured query language.*

Sauxe

Sauxe es un marco de trabajo que brinda solución a un sin número de escenarios o aspectos arquitectónicos como: gestión y configuración dinámica de cache, integración de componentes de forma distribuida o no distribuida, acceso a bases de datos a través de una capa de abstracción, gestión de concurrencia de recursos, administración centralizada de transacciones, gestión dinámica de las trazas generadas por los sistemas, implementación de mecanismos de autenticación y autorización, implementación de mecanismos de mensajería y control de excepciones, gestión y configuración dinámica de precondiciones, postcondiciones y validaciones de variables, gestión y configuración de flujos de trabajo, visualización de las funcionalidades de un sistema, entre otros escenarios de alta complejidad, garantizando los atributos de calidad de los sistemas que se desarrollen con el mismo. Cuenta con una arquitectura en capas que a su vez presenta en su capa superior un MVC (Baryolo y otros, 2010).

Strophejs

Es una colección de librerías para comunicarse con el protocolo XMPP, es usado para implementar juegos en tiempo real, sistemas de notificaciones, motores de búsquedas, así como para la mensajería instantánea tradicional. Es una librería JavaScript dado que este último no facilita conexiones del Protocolo de Control de Transmisión (TCP) persistentes, esta librería se basa en flujos bidireccionales sobre HTTP sincrónico (BOSH siglas en inglés) para emular la persistencia con estado de conexión en ambos sentidos a un servidor XMPP (Moffitt, 2010).

XMPPHP

XMPPHP es el sucesor de Class.Jabber.PHP que se ha estado elaborando durante años. Esta librería toma ventaja de PHP5, y proporciona una solución más completa, con un enfoque directo. Algunas de las características que incluye son:

- Permite conectarse a cualquier servidor XMPP 1.0 (Google Talk, Talk LJ, jabber.org, entre otros).
- Soporta el cifrado TLS.
- Varios enfoques de procesamiento XML y soporte de estilos (Werdmuller, 2010).

1.5.6 Herramientas de desarrollo

Servidor Web Apache v2.0

El Servidor HTTP Apache es un servidor web HTTP de código abierto para plataformas Unix, Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio

virtual. Cuando comenzó su desarrollo en 1995 se basó inicialmente en código del popular NCSA[9], pero más tarde fue reescrito por completo, el mismo consistía solamente en un conjunto de parches a aplicar al servidor de NCSA. Es una tecnología gratuita, y altamente configurable de diseño modular por lo que resulta muy sencillo ampliar sus capacidades. Actualmente existen muchos módulos para Apache que son adaptables a este, y están disponibles para su instalación cuando sean necesarios. Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor y es posible configurarlo para que ejecute un determinado script cuando esto suceda (Mohammed J, 2004).

Protocolo XMPP/Jabber.

El proyecto tiene definido como técnica de notificación en tiempo real el Protocolo XMPP/Jabber el cual es un protocolo estándar y abierto[10] basado en un conjunto de tecnología XML para aplicaciones en tiempo real, utilizando ficheros en formato XML para hacer las transferencias de mensajes entre sus entidades cliente-servidor o servidor-servidor. Las herramientas más conocidas que usan el protocolo XMPP son Jabber, Google Talk y las funciones de videoconferencia y audio conferencia de Google. Inicialmente fue desarrollado para implementar redes de mensajería instantánea en empresas (Petter Saint y otros, 2009).

Openfire v3.7.1

En el proyecto se tiene definido como servidor de mensajería instantánea Openfire el cual es un sistema de mensajería instantánea GPL y hecho en java. Utiliza el protocolo Jabber para servidor de mensajería permitiendo administrar usuarios, compartir archivos, auditar mensajes, mensajes fuera de línea (en inglés offline), grupos, etc. y además contiene plugins gratuitos con diferentes funciones extras. Utiliza SSL/TLS para la encriptación durante la transmisión de datos a través de las comunicaciones cliente-servidor o servidor-servidor. Es capaz de ofrecer estadísticas del servidor, mensajes y paquetes. Tiene soporte para la autenticación vía certificados y LDAP (Community, 2011).

El servidor de mensajería es configurable desde la Web, soporta grupos de usuarios, soporta conexión a otras redes de mensajería como MSN[11], GTalk[12], ICQ[13], etc. Esto permite estar conectado a varias redes desde un mismo cliente, Reporte de sesiones de usuarios, Soporte para

[9]**NCSA** (National Center for Supercomputing Applications): por sus siglas en español “Centro Nacional de Aplicaciones de Supercomputación”.

[10]**Abierto**: Con las ventajas del software libre, se puede ver el código.

[11]**MSN** (The MicroSoft Network): es una colección de servicios de internet ofrecidos por Microsoft.

[12]**GTalk** (Google Talk): es un cliente de mensajería instantánea desarrollado por Google.

[13]**ICQ**: es un cliente de mensajería instantánea.

Plugins14, Soporta LDAP, Soporta conexiones server-to-server para compartir usuarios (Landívar, 2009).

PostgreSQL 9.1

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado. PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando (DRAKE y otros, 2011).
A continuación se tienen algunas de las características más importantes y soportadas por PostgreSQL:
Generales: (DRAKE y otros, 2011).

- Es una base de datos 100% ACID[15].
- Integridad referencial.
- Replicación asincrónica/sincrónica / Streaming replication - Hot Standby.
- Copias de seguridad en caliente (Online/hot backups).
- Unicode.
- Juegos de caracteres internacionales.
- Múltiples métodos de autentificación.
- Acceso encriptado vía SSL.
- Actualización in-situ[16] integrada (pg_upgrade).
- Completa documentación.
- Licencia BSD.
- Disponible para Linux y UNIX en todas sus variantes y Windows 32/64bit.

Visual Paradigm8.0

[15] **ACID** (Atomicity, Consistency, Isolation and Durability): por sus siglas en español “Atomicidad, Consistencia, Aislamiento y Durabilidad”.
[16] **in-situ :** significa en el lugar.

Visual Paradigm es una herramienta CASE[17] la cual propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Constituye una herramienta privada disponible en varias ediciones, cada una destinada a unas necesidades: Enterprise, Professional, Community, Standard, Modeler y Personal. Existe una alternativa libre y gratuita de este software, la versión Visual Paradigm UML 8.0 Community Edition (la versión Community Edition, ya que existe la Enterprise, Professional, etc.). Fue diseñado para una amplia gama de usuarios interesados en la construcción de sistemas de software de forma fiable a través de la utilización de un enfoque Orientado a Objetos (Visual-Paradigm, 2013).

Se caracteriza por:

- Disponibilidad en múltiples plataformas (Windows, Linux).
- Diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo
- Licencia: gratuita y comercial.
- Soporta aplicaciones Web.
- Generación de código para Java y exportación como HTML.
- Soporte de UML versión 2.1.
- Interoperabilidad con modelos UML 2 (meta modelos UML 2.x para plataforma Eclipse) a través de XMI.
- Ingeniería inversa - Código a modelo, código a diagrama.
- Ingeniería inversa Java, C++, Esquemas XML, NET exe/dll, CORBA IDL.

[17]**CASE** (Computer Aided Software Engineering): por sus siglas en español “Ingeniería de Software Asistida por Computación”.

- Generación de código - Modelo a código, diagrama a código.
- Soporte ORM - Generación de objetos Java desde la base de datos.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) por sus siglas en inglés, existente a diagramas de Entidad-Relación.

Mozilla Firefox 17.0

Mozilla Firefox es un navegador web libre de código abierto, lo desarrolla una organización mundial sin ánimo de lucro que se dedica a promover el control por parte de los usuarios, la navegación es muy fácil por lo que se puede llegar rápidamente a los sitios favoritos de cualquier persona, tiene un alto rendimiento con tiempos de inicio más rápidos, aceleración de gráficos renderizados y mejoras en la velocidad de carga de las páginas, Firefox está lleno de grandes mejoras en el rendimiento, tiene un motor de JavaScript súper rápido, experimenta una aceleración gráfica rápida de vídeo y contenido web con un nuevo sistema gráfico basado en capas. Firefox se integra de un modo elegante con un programa antivirus. Cuando se descarga un archivo, el antivirus lo analiza automáticamente para proteger contra virus y otro software mal intencionado que podrían atacar el equipo. Firefox impide que los atacantes puedan interceptar la información sensible al establecer automáticamente conexiones seguras a los sitios web que ofrecen servidores HTTPS seguros (Mozilla, 2012).

NetBeans 7.3

NetBeans es un proyecto de código abierto. Sun MicroSystems fue el fundador de dicho proyecto en junio del 2000. Hoy en día hay disponibles dos productos: el NetBeans IDE y NetBeans Platform. NetBeans IDE es un entorno de desarrollo, una herramienta para que los programadores puedan implementar programas brindando opciones como ejecutar, compilar, depurar y otros. Está escrito en Java, pero es usado para el desarrollo de programas con otros lenguajes de programación. NetBeans IDE es un producto libre y gratuito sin restricciones de uso (Bock, 2012).

Control de VersionesRapidSVN 0.12.0

RapidSVN 0.12.0 es una plataforma visual para el sistema Subversion escrito en C++. Se utiliza un nuevo marco wxWidgets[18] y también se incluye un cliente de Subversion. El objetivo de este proyecto es conseguir un producto de fácil manejo para los usuarios principiantes pero lo

[18]**wxWidgets**: son unas bibliotecas multiplataforma y libres, para el desarrollo de interfaces gráficas programadas en lenguaje C++.

suficientemente potente y con herramientas interesantes para los usuarios avanzados. El programa funciona en cualquier plataforma y se puede ejecutar en Linux, Windows, Mac OS / X, Solaris, etc (Rapidsvn, 2013).

1.6 Conclusiones parciales del capítulo

Partiendo de los resultados obtenidos al haber realizado el estudio de las Reglas de Negocio, se decide para la solución que se presenta el uso de una regla de tipo restrictiva obligatoria, teniendo en cuenta que si no se cumple en su totalidad el componente desarrollado, no tiene la capacidad de notificar ningún evento. Las herramientas estudiadas no son convenientes utilizarlas porque no solucionan la problemática expuesta al inicio de la investigación. Se utilizará el modelo de desarrollo orientado a componentes que define el CEIGE, el cual provee durante el proceso de desarrollo, una guía que incluye las tareas, actividades, fundamentos y herramientas necesarias que ayuden a garantizar un software como resultado final de mejor calidad. Dentro de las tecnologías a utilizar se encuentran algunas definidas por el centro como, el servidor Apache y el gestor de bases de datos PostgreSQL, la herramienta de diseño Visual Paradigm, los lenguajes de programación PHP y JavaScript, los cuales son utilizados en los marcos de trabajo Zend Framework y EXT JS respectivamente, además del marco de trabajo Doctrine. El autor define además otras librerías que son necesarias para lograr la comunicación entre el servidor Openfire y el lenguaje PHP, es el caso de XMPPHP y Strophe para que el servidor de mensajería instantánea se pueda comunicar con el servidor apache y el lenguaje JavaScript respectivamente.

PROPUESTA DE SOLUCIÓN

2.1 Introducción

El siguiente capítulo muestra los resultados obtenidos durante el proceso de desarrollo de la solución para el componente de notificaciones basado en reglas de negocio, la cual comienza con la descripción del modelo conceptual, luego la identificación y descripción de los requisitos funcionales y no funcionales para entender mejor la solución, así como algunos artefactos generados por el modelo de desarrollo y que fueron obtenidos durante la fase de análisis y diseño del componente. Además se da una breve descripción de los patrones de diseños y arquitectónicos utilizados en la solución.

2.2 Propuesta de solución

La solución se enmarca en envíos de notificaciones mediante reglas de negocio, por lo que se define una regla de negocio de tipo restrictiva obligatoria, que va a estar comparando cada una de las acciones que están asociadas a un evento, con las peticiones que realiza cada usuario por su nivel de privilegio en cada uno de los subsistemas del marco de trabajo.

2.2.1 Definición de la regla

si (<petición> es igual <acción>) entonces <notificación>

Si los datos capturados (sistema, subsistema, acción, controlador) a las peticiones hechas por el usuario, coincide con la acción que está siendo invocada en ese subsistema, entonces se dispara el evento asociado a esa acción, y mediante un mensaje notifica al usuario los cambios realizados en determinado subsistema. Para lograr lo antes mencionado se debe seguir el siguiente orden para asociar los eventos a las acciones por subsistemas donde:

1. Se listan los subsistemas registrados y se selecciona uno.
2. Después se define el evento que se va a disparar.
3. Luego se escogen de los subsistemas subscritos las acciones para asociar el evento que se va a disparar.
4. Se asocia el evento a las acciones seleccionadas.

Al cumplirse dicha regla, permite que: un subsistema se le subscriba un evento definido, esto quiere decir que se van a disparar los eventos asociados a las acciones de un subsistemas, para notificar cada una de las acciones invocadas por los usuarios en uno de los subsistemas subscritos en el

marco de trabajo. Para lograr lo anteriormente planteado se determinó unir los módulos Gestionar notificaciones que emite y recibe cada subsistema en un solo módulo implementado para la gestión de las notificaciones. Dicho módulo se describe a continuación:

Módulo 1: Gestionar notificaciones por subsistema.

Gestiona las notificaciones que emite y recibe cada subsistema.

2.3 Modelo conceptual.

En la Figura 2 se muestra modelo conceptual, en el mismo se define que Sauxe tiene varios subsistemas que puedes ser emisores o receptores, estos ejecutan múltiples acciones que son las que disparan uno o varios eventos y además definen múltiples eventos que estos se subscriben al subsistema si se cumple la regla de negocio global, y uno o varios eventos que están subscritos al subsistema también pueden ejecutar una o varias acciones.

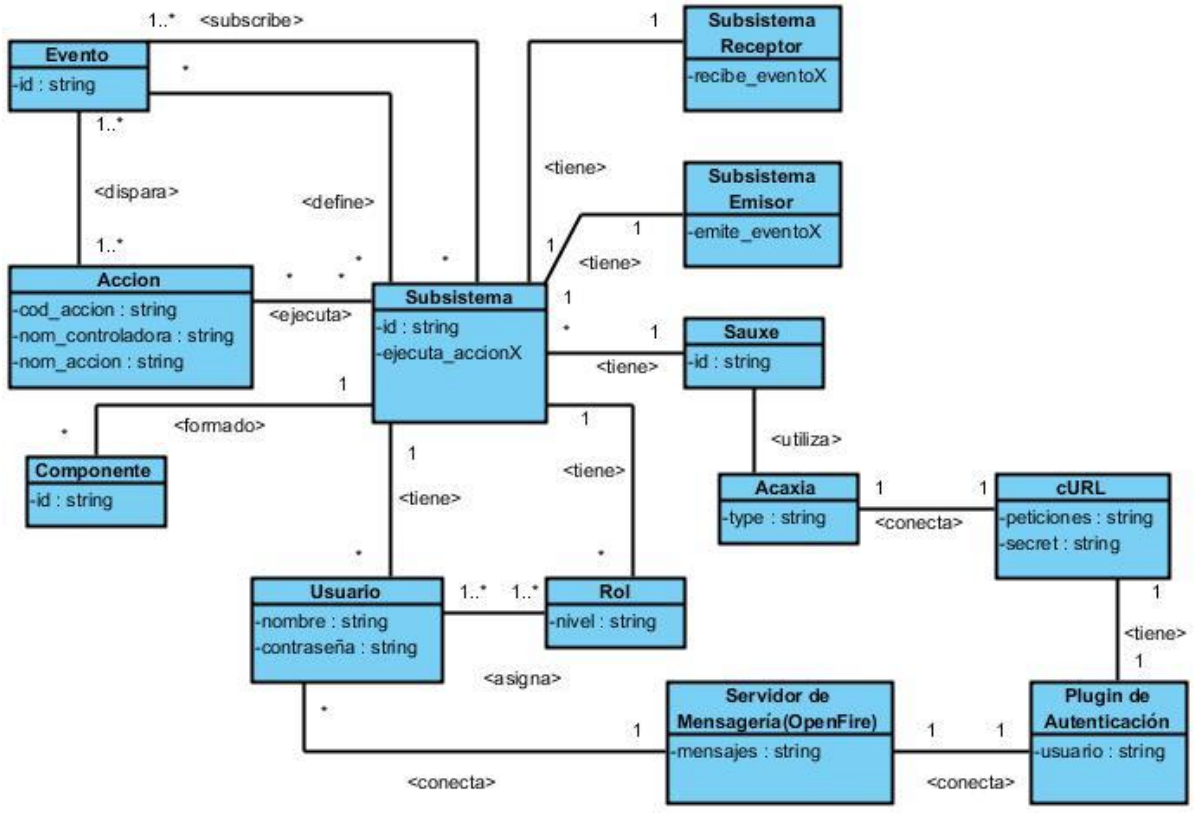


Figura 2. Modelo conceptual (Fuente: Realización propia)

2.4 Requisitos de software

El proceso de desarrollo de software comprende en sus etapas tempranas tareas orientadas a captar las necesidades o características a satisfacer en el sistema que se vaya a crear o modificar. Como resultado de esta captación se obtendrán los requisitos con los que deberá cumplir la

solución, que pueden variar a lo largo del ciclo de vida del sistema. Si se realiza un buen proceso de captura de requisitos, el resultado será requisitos claros, completos y consistentes; de lo contrario es probable que se construya una solución que resuelva incorrectamente el problema, obteniendo pérdidas en tiempo y costes, frustración personal y clientes descontentos. El proceso de definición de los requisitos funcionales es dirigido por el equipo de desarrollo teniendo en cuenta la información brindada por el cliente que realiza dicho proceso. Posteriormente a partir de la información que brinda el cliente se elaboran los documentos de especificación de requisitos y finalmente se valora y valida la información en busca de errores, inconsistencias o faltas para evitar u omitir algún requerimiento del cliente (Pressman, 2002).

2.4.1 Técnicas de captura y validación de requisitos utilizadas

Durante el proceso de ingeniería de requisitos se proponen como técnicas de captura de requisitos las siguientes: utilización de **mapas conceptuales** para la representación gráfica de las ideas y sus relaciones, **entrevistas** con el cliente para conocer sus exigencias con mayor claridad y sentar las bases de la captura de requisitos, **tormentas de ideas** para realizar talleres con el equipo de desarrollo y los tutores con el objetivo de debatir el tema, y la realización de **reuniones** con la participación del equipo de desarrollo para refinar las ideas que puedan ser difíciles de traer a la superficie usando entrevistas. Mediante la aplicación de estas técnicas mencionadas anteriormente se pudo obtener los requisitos de software del componente de gestión de notificaciones basado en reglas de negocio, quedando identificados 12 Requisitos Funcionales (RF) con los que debe cumplir el componente, los cuales se listan a continuación:

- **RF1 El sistema debe permitir gestionar notificaciones por subsistema.**
- RF1.1 Adicionar notificación emitida.
- RF1.2 Modificar notificación emitida.
- RF1.3 Eliminar notificación emitida.
- RF1.4 Listar las notificaciones emitidas.
- RF1.5 Adicionar notificación recibida.
- RF1.6 Eliminar notificación recibida.
- RF1.7 Modificar notificación recibida.
- RF1.8 Listar notificaciones recibidas.

- RF1.9 Listar subsistemas.
- RF1.10 Listar acciones.

Teniendo en cuenta la importancia del proceso de Validación de Requisitos se propone utilizar varias técnicas de estas como son: la **Revisión Técnica Formal** en las que estén presentes la gran mayoría del personal involucrado en la solución, tanto por parte del cliente como de los desarrolladores, en la revisión formal el equipo de desarrollo conduce al cliente a través de los requerimientos, explicándole las implicaciones de cada uno, además se propone la creación de los **Prototipos de Interfaz de Usuario** para hacer una representación aproximada de la interfaz de usuario de un sistema software que permita al cliente entender fácilmente la propuesta de los requisitos para resolver sus problemas de negocio y la **Generación de casos de prueba** para probar cada requisito funcional identificado. A continuación se muestra la descripción del requisito funcional Adicionar notificaciones emitidas, las demás descripciones de requisitos funcionales se pueden apreciar en el anexo 2.

2.4.2 Especificación de Requisitos

RF1: Gestionar notificaciones.

RF1.1: Adicionar notificación emitida

Precondiciones:	El usuario debe estar autenticado en el sistema y debe poseer permisos para ejecutar esta acción.
Descripción	
Adiciona a un subsistema la notificación (código y mensaje) que emite.	
Flujo de eventos	
Flujo básico de eventos	
1 El usuario Selecciona la interfaz Gestionar notificaciones.	
2 Seleccionar un sistema.	
3 Seleccionar la opción Adicionar en la interfaz.	
4 Introduce los datos.	
5 Escoger la opción Aceptar.	
Post-Condiciones	
1 El sistema muestra un mensaje: ¨Se adicionó la notificación satisfactoriamente¨.	
Validaciones	
Se validan los datos según lo establecido en el Modelo conceptual.	
Conceptos	

Visible en la interfaz: sistema, código y mensaje.

Tabla 1. Descripción del requisitos funcional Adicionar notificación emitida.

Prototipo elemental de interfaz gráfica de usuario.

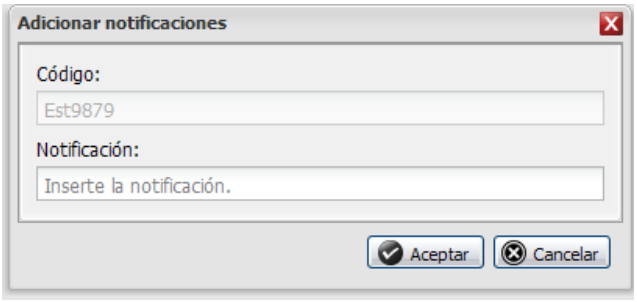


Figura 3. Interfaz gráfica del requisito Adicionar notificación emitida.

2.4.3 Requisitos no funcionales

Los requerimientos no funcionales forman una parte significativa de la especificación. Estos son importantes para que clientes y usuarios puedan valorar las características no funcionales del producto, pues si se conoce que el mismo cumple con toda la funcionalidad requerida, entonces, las propiedades no funcionales, como cuán usable, seguro, conveniente y agradable, pueden marcar la diferencia entre un producto bien aceptado y uno con poca aceptación. Como se ha mencionado con anterioridad, el presente trabajo forma parte de un proceso productivo iniciado por el CEIGE y los resultados que se obtengan formarán parte del marco de trabajo desarrollado en el mismo. Por tanto los requisitos no funcionales con los que debe cumplir la aplicación a desarrollar fueron establecidos por el centro al inicio del proceso de desarrollo, a continuación se describen los más importantes (González, 2011).

2.4.3.1 Software

Para el cliente:

- Navegador Mozilla Firefox 3.0 o superior.
- Sistema operativo Windows 98 o superior o Linux.

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.

2.4.3.2 Rendimiento

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

2.4.3.3 Seguridad

Autenticación y autorización (Contraseña de acceso), ya que el sistema podrá ser utilizado solamente por usuarios autenticados en el mismo, mostrándole así las notificaciones correspondientes. 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.

2.4.3.4 Hardware

Para el servidor Openfire:

- 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.

Para el cliente:

- Requerimientos mínimos: Procesador Pentium III a 1GHz con 256Mb de memoria RAM. Sistema operativo Windows 98 o superior o Linux.
- Tarjeta de red.

2.5 Arquitectura de datos

2.5.1 Modelo de datos

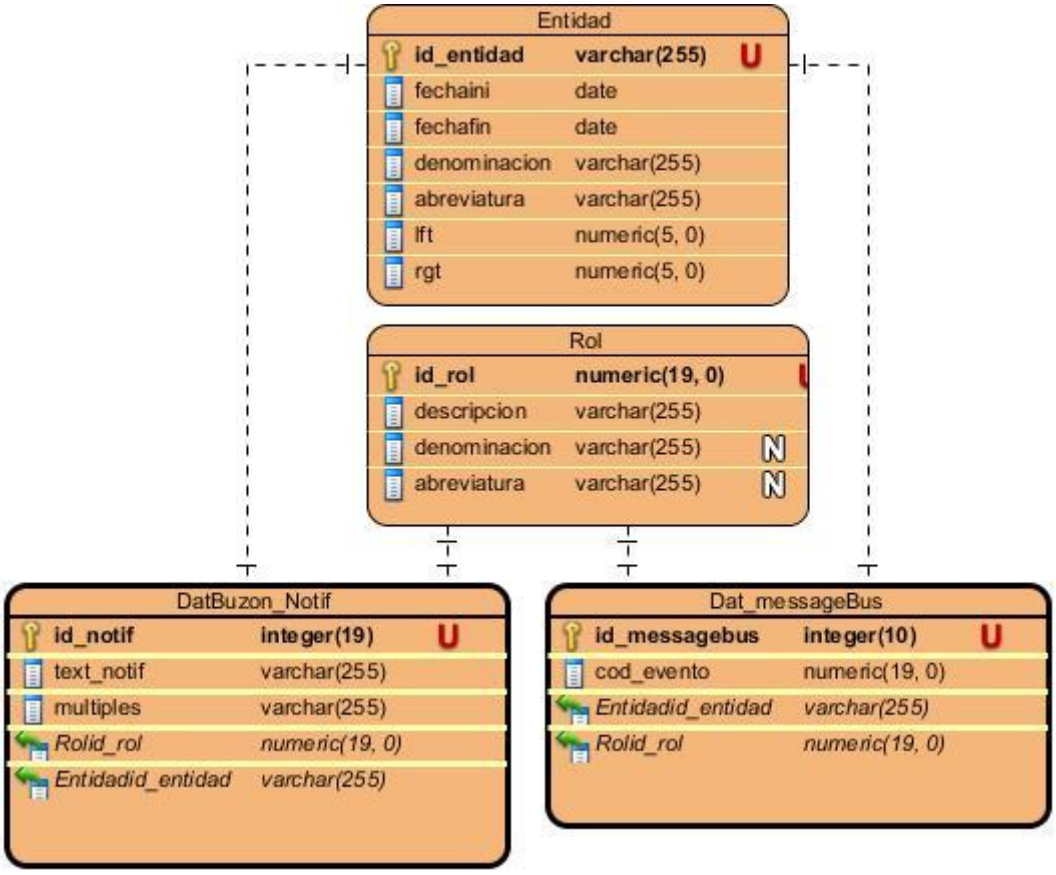


Figura 4. Modelo de datos.

El modelo de datos propuesto en la solución cuenta con un total de 2 tablas. La tabla **DatBuzon_Notif** almacena todas las notificaciones emitidas por los subsistemas registrando el rol que la recibe y la entidad a la cual pertenece dicho rol, además es la fuente de datos cuando un usuario pide listar todas las notificaciones que ha recibido. La tabla **Dat_messageBus** almacena la información que tiene que ver con los roles que tiene asociado un evento en una entidad, registrando el rol que la recibe y la entidad a la cual pertenece dicho rol.

2.6 Diagrama de componente

"El diagrama de componentes muestra la organización y dependencias entre los componentes (parte modular, desplegable y reemplazable de un sistema que encapsula una implementación y expone una serie de interfaces)" (Pressman, 2002). El diagrama de componentes es usado para estructurar el modelo de implementación y mostrar las relaciones entre los elementos de este, proporcionando las interfaces necesarias para la utilización de sus funcionalidades.

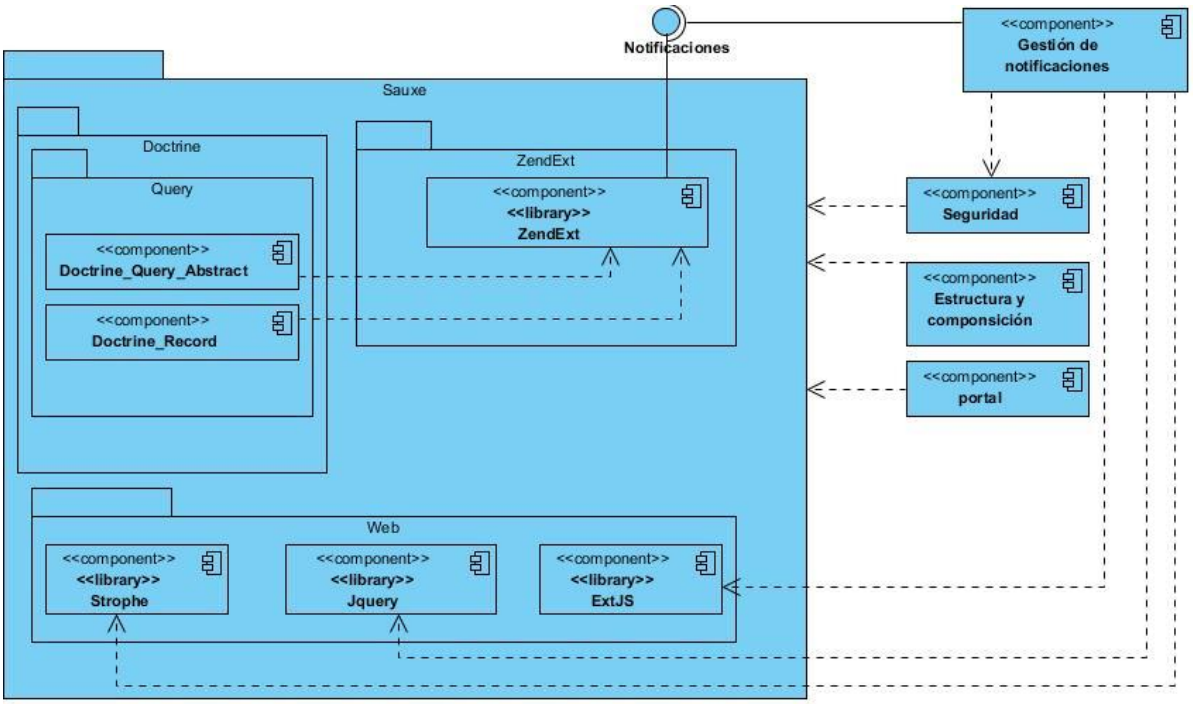


Figura 5. Diagrama de componentes.

2.7 Diagrama de clases del diseño

En la Figura 7 se muestra el diagrama de clases del diseño de los escenarios gestionar notificaciones, donde se representan las principales clases, operaciones y relaciones que se necesitan para darle cumplimiento a los requisitos funcionales relacionados con la gestión de las notificaciones. Las clases gestnotificaciones.js y Gestnotificaciones.phtml conforman la capa arquitectónica de presentación. La clase GestNotificacionesController solo maneja la comunicación entre la vista y el modelo, la clase GestNotificacionesModel es la encargada de la lógica del negocio, implementando funcionalidades que garantizan el cumplimiento de los requisitos identificados, manejando los archivos xml XML_subsemitennotif y XML_subsrecibennotif. En el fichero subemitennotif.xml se almacena cada subsistema con las notificaciones que emiten y en el fichero subrecibennotif.xml se almacena cada subsistema con las notificaciones que reciben. Por otro lado el sistema en ZendExt_Controller_Secure tiene un controlador de acciones personalizado e integrado a seguridad, el cual contiene todas las acciones para que los aspectos asociados a la acción sean disparados en el momento que suceden en el marco de trabajo para posteriormente mostrarlas según el evento que se subscribe al subsistema.

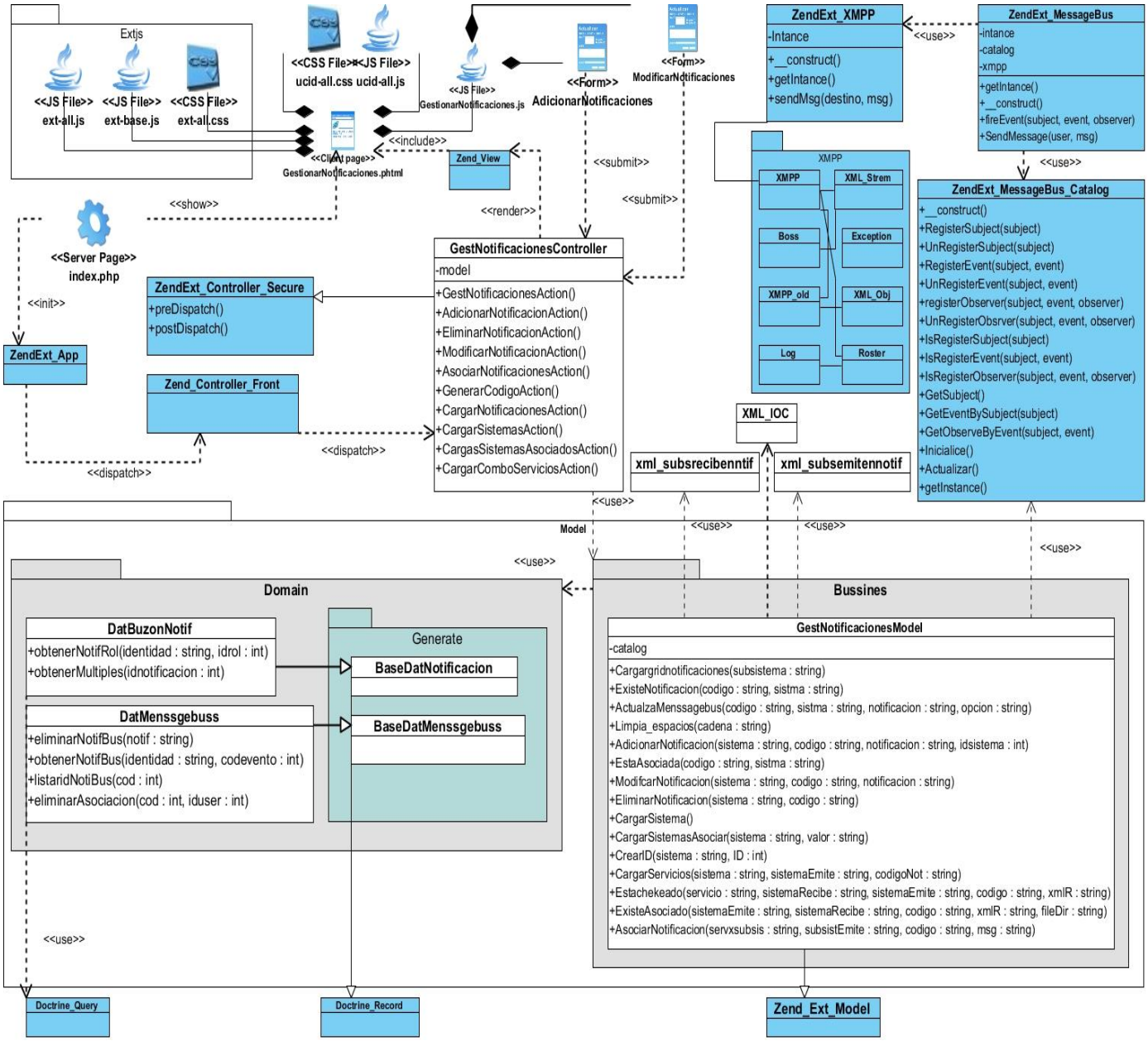


Figura 6. Diagrama de clases del diseño del módulo. Gestionar Notificaciones por subsistema.

2.7.1 Descripción de las clases comunes para todos los diagramas de clases del diseño

* Se refiere al nombre de las clases de los diferentes diagramas. Ejemplo *.js hace referencia a todas las clases con extensión .js.

Clase	***Descripción***
***.js**	Este java script se encarga de crear todas las interfaces que serán usadas por las diferentes funcionalidades.

***.phtml**	En este HTML se incluyen todos los java script para mostrar al usuario las interfaces que serán utilizadas en las diferentes funcionalidades.
index.php	Se encarga de determinar cuál es la clase controladora que le corresponde a cada página cliente.
<<Framework>>ExtJs	Librería JavaScript que utiliza la clase gestnotificacion.phtml para gestionar las notificaciones de cada subsistema.
Paquete UCID	Paquete JavaScript que utiliza la clase gestnotificacion.phtml para realizar las notificaciones en los datos introducidos por el usuario al igual que para crear las interfaces.
***Controller**	Es la clase que gestiona la lógica de negocio y se encarga de capturar los parámetros que le son enviados desde la interfaz y devolver a las interfaces la información solicitada por el usuario.
***Model**	Es la clase encargada de procesar el negocio e interactuar con los modelos. Adiciona, modifica y elimina en la tabla dat_BuzonNotif.
Base*	Es la clase base la cual tiene mapeado todos los campos de una tabla en la base de datos.
ZendExt_Controler_Secure	Clase padre del frameworkZend de la cual extienden las clases controladoras.
ZenExt_Model	Entre las características de esta clase se pueden mencionar que obtiene las conexiones activas en el sistema cuando se trata de acceder a una clase determinada del modelo.
Zend_View	Esta clase trabaja con la visión del patrón modelo-vista-controlador. Existe para ayudar a mantener el script de la vista separado del modelo y scripts del controlador.
Zend_Controller_Front	Es una interpretación del patrón FrontController, procesa todas las solicitudes recibidas por el servidor y es responsable en última instancia de la delegación de las solicitudes a los ActionControllers.

2.8 Diagrama de secuencia

A continuación se muestra el diagrama de secuencia del Adicionar notificación que emiten subsistemas el resto se pueden apreciar en el anexo 3.

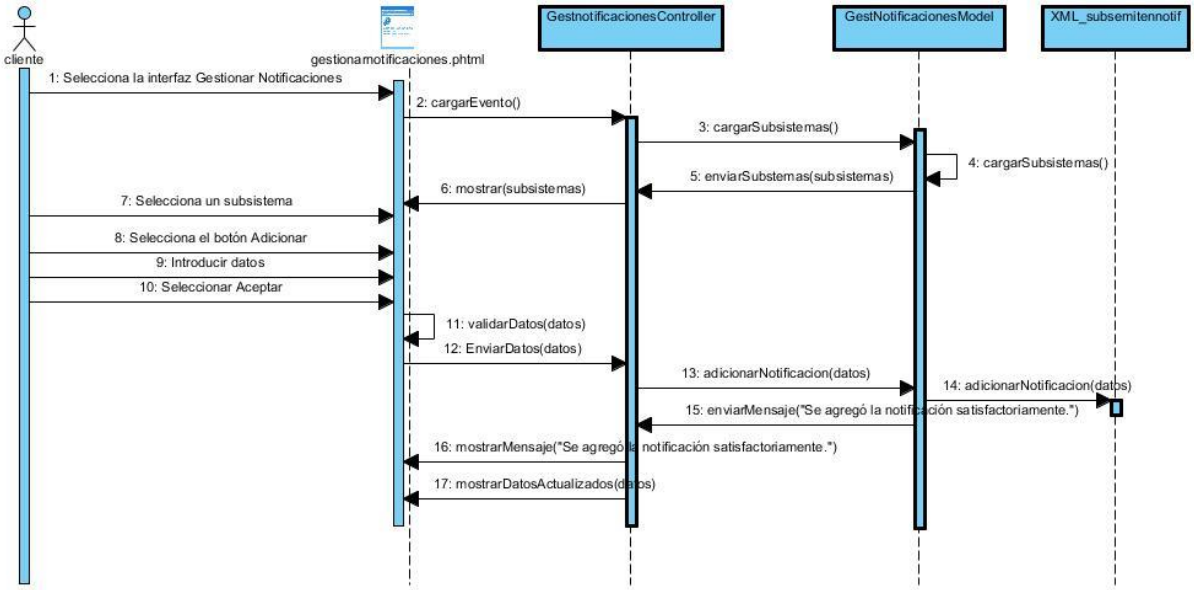


Figura 7 Diagrama de secuencia. Adicionar notificación que emiten subsistemas.

2.9 Patrones de Diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software, por lo que es de suma importancia aplicarlos en la construcción del diseño de un sistema. Con su uso, se pretende establecer un lenguaje común entre los programadores, contribuir a la reutilización, ahorrar tiempo en la implementación y obtener un producto con calidad. Son además la solución a determinado problema de diseño que se presente en el desarrollo de un sistema. Entre sus características debe incluir la habilidad de ser reutilizable y aplicable a diferentes problemas de diseño en distintas circunstancias.

Brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Se deben tener presente los siguientes elementos de un patrón: su nombre, el problema (cuándo aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios) (Tedeschi, 2012).

2.9.1 Patrones GoF

Los patrones Gang Of Four (GoF) por sus siglas en inglés, son patrones de diseño publicados en el libro DesignPatterns: Elements of Reusable Object-Oriented Software por Gamma, Helm, Jonson y Vlissides conocidos por Banda de los cuatro. Están divididos fundamentalmente en tres grandes grupos:

- **Creacionales:** Concierne al proceso de creación de objetos.
- **Estructurales:** Tratan la composición de clases y/o objetos.
- **De Comportamiento:** Caracterizan las formas en las que interactúan y reparten responsabilidades las distintas clases u objetos.

De los patrones GoF existentes a continuación se describen los que fueron utilizados para el desarrollo del componente y de qué forma beneficiaron al mismo:

Patrón Singleton:

El patrón Singleton o Solitario garantiza que una clase sólo tenga una instancia única y permite que la misma sea accedida desde cualquier parte del sistema. Es utilizado en las clases: ZendExt_Xmpp, ZendExt_MessageBus y ZendExt_MessageBus_Catalog. Ninguna de estas clases puede tener más de una instancia de ellas mismas, porque de no ser así, podría haber contradicción entre los datos que manejan.

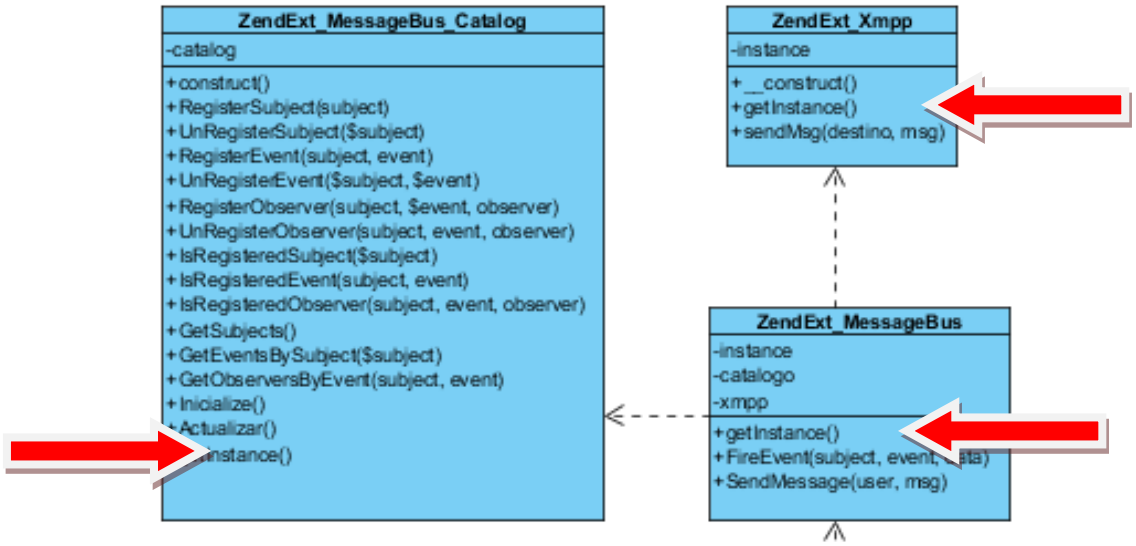


Figura 8 Patrón Singleton presente en el componente

Patrón Fachada:

El patrón estructural Fachada mediante una única interfaz se encarga de relacionar varias clases de un subsistema, para facilitar su uso. En el componente de gestión de notificaciones a implementar se usa patrón fachada, específicamente la clase ZendExt_Xmpp que utiliza funcionalidades de la librería XMPPHP y contribuye a que existan puntos de acceso fáciles para utilizar la librería que conecta el lenguaje PHP con el protocolo XMPP.

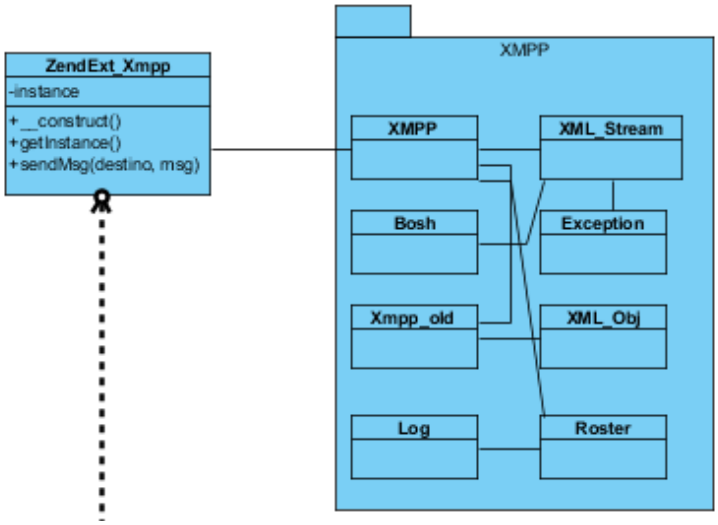


Figura 9 Patrón Fachada presente en el componente.

2.9.2 Patrones GRASP

Experto: Propone que la clase que contenga toda la información necesaria, será la responsable de la creación de un objeto o la implementación de un método. Se puede ver el uso de este patrón en todas las clases a utilizar en la solución ya que cada clase conoce su información y es la encargada de implementar las funcionalidades que brindan información requerida de las mismas, siendo más fáciles de entender, mantener y ampliar, aumentando sus posibilidades de reutilización.

Creador: Plantea asignarle a la clase B la responsabilidad de crear una instancia de la clase A. La clase controladora GestnotificacionesController se encarga de la creación de las instancias de las clases que describen los objetos que en ellos se manejan, ya que estos o agregan, contienen, registran o utilizan las instancias de estos objetos. Esto favorece la reutilización y el bajo acoplamiento.

Controlador: El patrón controlador funciona como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la interfaz quien recibe los datos del usuario y los envía a las distintas clases según el método invocado. Ejemplo de esto es la clase controladora GestnotificacionesController, que se encarga de llevar el control de todas las notificaciones relacionados con el negocio e implementa las funcionalidades que dan respuesta a las peticiones del usuario.

Bajo acoplamiento: Este patrón expresa que entre las clases deberán existir pocas ataduras, es decir, estas estarán lo menos relacionadas posibles. El uso de éste patrón se evidencia en la poca

relación existente entre las clases que conforman el módulo. Esto favorece la reutilización, y el mantenimiento futuro del componente.

Alta cohesión: Propone que la información que almacena una clase debe de ser coherente y debe estar, en la medida de lo posible, relacionada con la clase. Ejemplo de esto es el controlador GestnotificacionesController el cual al recibir un mínimo cambio, requiere una completa restructuración de los métodos para su correcto funcionamiento. Esto incrementa la claridad, la reutilización y la facilidad de comprensión del diseño.

2.10 Estilo Arquitectónico

El estilo arquitectónico definido por el Departamento de Tecnología, donde se desarrolla el marco de trabajo Sauxe, es el **Estilo N capas**. Se identificaron 3 niveles o capas fundamentales, como se describe a continuación:

- **Capa Datos:** Donde estarán ubicados los servidores de base de datos PostgreSQL y además, un conjunto de Ficheros de Configuración del sistema.

- **Capa de Acceso a Datos:** Donde estará presente el framework Doctrine, como persistidor para la comunicación con el servidor de datos mediante el protocolo PDO[19], también estará un persistidor de configuración que es el encargado de comunicarse vía XML con los Ficheros de Configuración del sistema, y el último elemento de esta capa es la Fachada de Transferencia de Acceso Rápido cuya función es el acceso rápido tanto a los ficheros de comunicación como al servidor de datos, a través del Persistidor de Configuración y Doctrine para cada caso en específico.

- **Capa de Funcionamiento:** En esta capa se emplea el patrón de arquitectura Modelo Vista Controlador (MVC), así como el elemento de Acceso de Respuesta Rápida el cual brinda informaciones de forma vertiginosa. De forma vertical al modelo descrito hasta este momento, estará el módulo de cacheo del sistema así como el encargado del tratamiento de la seguridad a nivel de aplicación web.

[19]**PDO: P**rocess **D**ata **O**bjects. Protocolo de comunicación especialmente adecuado para la transmisión rápida de datos; este modelo define un productor y uno o varios consumidores PDO.

Esta gran estructura descrita se comunica con una serie de servicios web mediante protocolos SOAP[20], REST[21], HTTP[22] y HTTPS[23,] que interactúan con el sistema proporcionándole un conjunto de funcionalidades tanto de seguridad como de negocio.

2.11 Patrones Arquitectónicos.

Patrón arquitectónico Modelo-Vista-Controlador (MVC):

Como se expuso en el epígrafe anterior, el patrón arquitectónico Modelo-Vista-Controlador (MVC) se emplea en la capa de funcionamiento y es el encargado de separar los datos de la aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos:

- **Modelo:** Está compuesto por datos, reglas de negocio y las funcionalidades correspondientes para la comunicación con el persistidor de datos Doctrine, por ejemplo la clase GestNotificacionesModel.php del módulo Gestionar notificaciones.

- **Vista:** Muestra la información del modelo al usuario, por ejemplo la clase gestnotificaciones.phtml del módulo Gestionar notificaciones.

- **Controlador:** Gestiona las entradas y las respuestas del sistema al usuario, por ejemplo la clase GestnotificacionesController.php del módulo Gestionar notificaciones.

2.12 Conclusiones Parciales

Se concluye este capítulo dejando claro los requerimientos funcionales que debe cumplir el componente, los cuales a través de las técnicas de captura y validación de los mismos y con la ayuda del modelo conceptual permiten tener una idea y alcance del componente. Con este fin también se elaboraron los diagramas de clases del diseño en los que se plantean las clases a implementar y la relación entre ellas incluyendo los patrones arquitectónicos y de diseño propuestos. Con todos estos elementos y en conjunto con el modelo de datos físicos, se sientan las bases para la próxima fase, donde se implementarán y se le realizarán las pruebas de software al componente propuesto.

[20]**SOAP: S**imple **O**bject **A**ccess **P**rotocol. Permite la comunicación entre aplicaciones a través de mensajes por medio de Internet.
[21]**REST**: **R**epresentational **S**tate **T**ransfer o Transferencia Representacional de Estados. Sirve para acceder a servicios sencillamente a través de URLs.
[22]**HTTP: H**ypertext **T**ransfer **P**rotocol o Protocolo de Transferencia de Hipertexto. Protocolo cliente-servidor que articula los intercambios de información entre los clientes web y los servidores HTTP.
[23]**HTTPS:** Versión segura del protocolo **HTTP** que implementa un canal de comunicación seguro y basado en **SSL** (Secure Socket Layers) entre el navegador del cliente y el servidor HTTP.

IMPLEMENTACIÓN Y PRUEBA

3.1 Introducción

En el siguiente capítulo se aborda los estándares de codificación empleados durante la implementación del componente de notificación, las métricas que se utilizaron para la evaluación de la solución, los estándares de codificación definidos para la implementación. También lo concerniente a los artefactos generados por el modelo de desarrollo utilizado, específicamente el diagrama de despliegue y algunas pruebas efectuadas al componente para validar la solución del mismo.

3.2 Estándares de codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. El uso de estándares de codificación permite lograr un código más legible y reutilizable, de tal forma que se pueda aumentar su mantenibilidad a lo largo del tiempo (Pérez Alfonso, 2012).

Para el desarrollo del componente de gestión de notificaciones basado en reglas de negocio, se utilizarán algunos de los estándares de codificación y normas propuestas como parte de la línea de la arquitectura determinada para el desarrollo del ERP Cuba (Pérez Alfonso, 2012).

Los estándares utilizados en la codificación fueron los siguientes:

- **Notación PascalCasing:** Es como la notación húngara pero sin prefijos. En este caso los identificadores y nombres de las variables, métodos y funciones están compuestos por múltiples palabras juntas, iniciando cada palabra con letra mayúscula.

- **Notación CamelCasing:** Es parecido al PascalCasing con la excepción que la letra inicial del identificador no debe estar en mayúscula.

3.2.1 Nomenclatura de las clases

Los nombres de las clases comienzan con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación PascalCasing. Con sólo leerlo se reconoce el propósito de la misma.

Ejemplo: DatServidor.

3.2.2 Nomenclatura según el tipo de clases

1. **Clase controladora.**

Las clases controladoras después del nombre llevan la palabra: “Controller”.

Ejemplo: GestnomgestorController.

2. **Clases de los modelos.**

- **Business (Negocio).**

Las clases que se encuentran dentro de Business después del nombre llevan la palabra: “Model”.

Ejemplo: DatGestorModel

- **Domain (Dominio).**

Las clases que se encuentran dentro de Domain el nombre que reciben es el de la tabla en la Base de Datos.

Ejemplo: DatGestor

- **Generated (Dominio bases)**

Las clases que se encuentran dentro de Generated el nombre comienza con la palabra: “Base” y seguido el nombre de la tabla en la Base de Datos.

Ejemplo: BaseDatGestor.

3.2.3 Nomenclatura de las funciones

El nombre a emplear para las funciones se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, y en caso de ser una acción de la clase controladora se debe especificar el nombre de dicha acción en minúscula y seguido del sufijo “Action”.

Ejemplo:modificargestorAction.

3.2.4 Nomenclatura de las variables

El nombre a emplear para las variables se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, y comenzando con un prefijo según el tipo de datos.

Ejemplo: $gestor.

Prefijos para los tipos de datos

Los prefijos a utilizar en la creación de variables serán los siguientes:

Tipos de Datos	*Prefijos*
Arreglos	arr
Objetos	obj
Enteros	int
Cadena	str
float	flt
Boolean	Boo

Tabla 2 Prefijos a utilizar en la creación de variables.

3.2.5 Normas de comentarios

Se debe comentar todo lo que se haga dentro del desarrollo, establecer las pautas que conlleven a lograr un código más legible y reutilizable y así se pueda aumentar su mantenimiento a lo largo del tiempo.

Nomenclatura de los comentarios

Los comentarios deben ser lo bastante claros y precisos de forma tal que se entienda el propósito de lo que se está desarrollando.

1. **En las clases**

Antes de la declaración de una clase se escribe una breve descripción donde se explique el propósito de la misma. Se escribe de la siguiente forma:

```
/**
* Nombre de la clase *
* Descripcion *
* @author *
```

* @package *(módulo)

* @subpackage *(sub módulo)

* @copyright *

* @version (versión - parche) */

2. En las funciones

Antes de la declaración de la función se escribe una breve descripción donde se explique el propósito de la misma. Se escribe de la siguiente forma:

/**

* Nombre de la función *

* Descripcion *

* @author * (en caso de que no sea el autor de la clase)

* @param *(los parámetros que se le pasan a la función con su descripción)

* @throws *(en caso de que dispare una excepción)

* @return *(se pone lo que devuelve la función y un comentario)

*/

3.2.6 Estilo del código

En la implementación cuando se escriba una sentencia en php la forma de utilizar los tab del mismo es la siguiente:

```
<?php
//código
?>
```

Figura 10. Estilo del código.

- **Sangría o Indexado**

La política de sangría a utilizar en la implementación es por **tab.**

Ejemplo:

```php
<?php
/**
 * Indentation
 */
class Example {
    var $theInt = 1;
    function foo($a, $b) {
    switch ( $a) {
    case 0 :
    $Other->doFoo ();
    break;
    default :
    $Other->doBaz ();
    }
    }
    function bar($v) {
    for($i = 0; $i < 10; $i ++) {
    $v->add ( $i );
    }
    }
}
?>
```

Figura 11 Estilo del código: Sangría o Indexado.

- **Brazas o llaves.**

En la declaración de clases o interfaces, métodos, bloques y switch, la apertura de llaves se hace en la misma línea.

Ejemplo:

```php
<?php
/**
 * Braces
 */
interface EmptyInterface {
}

class Example {
    function bar($p) {
        for($i = 0; $i < 10; $i ++) {
        }
        switch ( $p) {
            case 0 :
                $fField->set ( 0 );
            break;
            case 1 :
                {
                    break;
                }
            default :
                $fField->reset ();
        }
    }
}
?>
```

Figura 12. Estilo del código: Brazas o llaves.

- **Espacios en blanco.**

1. **Arreglos.**

La declaración de los espacios en blanco en los arreglos es como se muestra en el ejemplo.

Ejemplo:

```php
<?php
class MyClass implements IO, I1, I2 {
}
class MyClass {
    public $a = 0, $b = 1, $c = 2, $d = 3;
    const MY_TRUE = 1, MY_FALSE = 2;
}
function foo() {
}
function bar(int $x, $y, $z = 1) {
}
?>
```

Figura 13. Estilo del código: Espacios en blanco en los Arreglos (Pérez Alfonso, 2012).

3.3 Métricas de software

Las métricas del producto son una medida cuantitativa que permite a las personas que tienen que ver con el software tener una visión profunda de la eficacia de los procesos del software y de los proyectos que dirigen. Se reúnen los datos básicos de calidad y productividad que son analizados, comparados y evaluados para determinar las mejoras en la calidad y productividad. Las métricas pueden ser usadas para señalar áreas con problemas a través de una evaluación objetiva, de manera que puedan ser solucionados y de esta forma mejorar el proceso de desarrollo del software. Existen cuatro razones para medir los procesos del software, los productos y los recursos, Caracterizar, Evaluar, Predecir y Mejorar (Pressman, 2002).

- Se caracteriza para comprender mejor los procesos, los productos, los recursos y los entornos de trabajo para establecer las líneas bases para las comparaciones con evaluaciones futuras.

- Se evalúa para determinar el estado con respecto al diseño, las medidas sirven como sensores para saber cuándo nuestros proyectos y procesos no se encuentran bajo control.

- Se predice para poder planificar y aumentar la comprensión de las relaciones entre los procesos y los productos y así establecer objetivos alcanzables para el coste, planificación y calidad de manera que se puedan aplicar los recursos apropiados.

- Se mide para mejorar la recogida de información cuantitativa, que nos ayuda a identificar los problemas, ineficiencias y oportunidades para mejorar la calidad del producto y el rendimiento del proceso.

Las métricas empleadas para evaluar los atributos de calidad fueron: Tamaño operacional de clase (TOC) y Relaciones entre clases (RC). En el momento de escoger ambas métricas, se partió del hecho fundamental: que el componente realizado, se basa en la Programación Orientada a Objetos (POO). Donde la clase representa una unidad básica y fundamental, por esta razón resulta idóneo utilizar atributos medibles que tengan como objetivo y eje central las clases, los cuales permiten: caracterizar, evaluar, predecir y mejorar el proceso de desarrollo (Pressman, 2002). Los atributos a evaluar son los siguientes:

- **Responsabilidad:** Se le asigna a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.

- **Complejidad de implementación:** Grado de dificultad en la implementación de un diseño de clases determinado.

- **Reutilización:** Nivel de reutilización que tiene una clase o estructura de clase, dentro de un diseño de software determinado.

- **Acoplamiento:** Valor de dependencia de una clase o estructura de clase con otras. Este atributo está muy ligado al de reutilización.

- **Complejidad del mantenimiento:** Categoría de esfuerzo para realizar un arreglo, mejora o 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:** Número de esfuerzos para realizar las pruebas de calidad (Unidad) del producto (componente, clase, conjunto de clases, etc.) diseñado (González, 2011).

3.3.1 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.

Tabla 3 Atributos de calidad evaluados por la métrica TOC.

Para estos atributos de calidad están definidos los siguientes criterios y categorías de evaluación:

Atributo	Categoría	Criterio
Responsabilidad.	Baja	<=Promedio
	Media	Entre promedio y 2 * Promedio
	Alta	>2*Promedio
Complejidad implementación.	Baja	<=Promedio
	Media	Entre Promedio y 2 * Promedio
	Alta	>2*Promedio
Reutilización.	Baja	>2*Promedio
	Media	Entre Promedio y 2 * Promedio
	Alta	<=Promedio

Tabla 4 Criterios de evaluación de la métrica TOC.

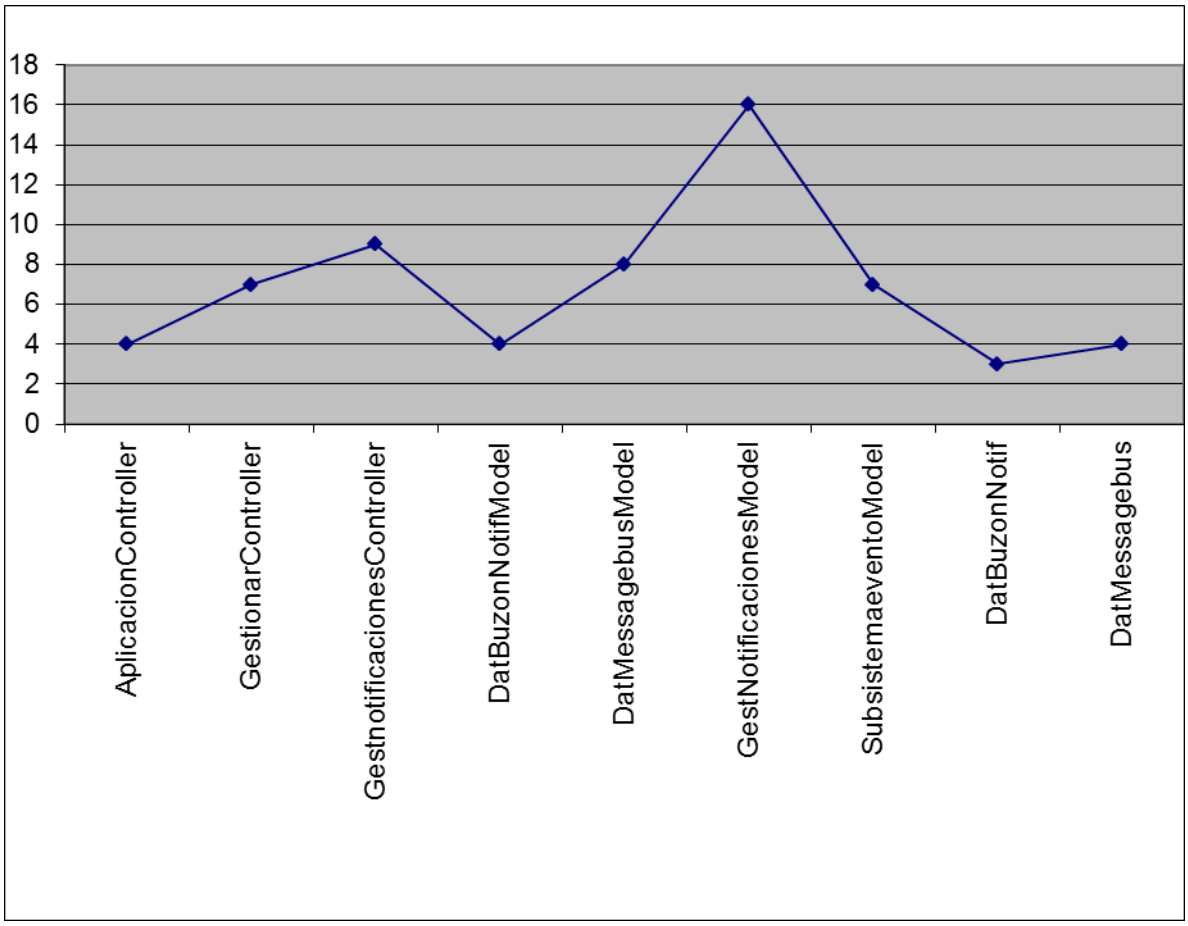
3.3.2 Resultados obtenidos al aplicar (TOC)

En la siguiente tabla se recoge como datos las 9 clases a las cuales se le aplicó la métrica TOC, en la columna cantidad de procedimientos se guardan un total de 62 métodos con los atributos de calidad que utiliza la métrica, categorizados en baja, media o alta según los criterios que aparecen en la tabla anterior, para la cual se calculó el promedio en la siguiente tabla arrojando un valor de 6.88.

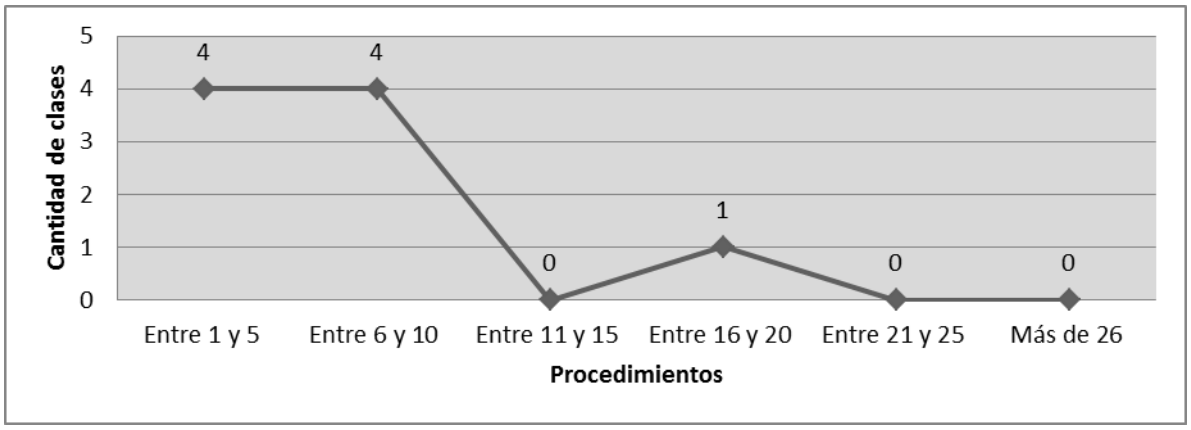
Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
AplicacionController	4	Baja	Baja	Alta
GestionarController	7	Baja	Baja	Alta
GestnotificacionesController	9	Media	Media	Media
DatBuzonNotifModel	4	Baja	Baja	Alta
DatMessagebusModel	8	Media	Media	Media
GestNotificacionesModel	16	Alta	Alta	Baja
SubsistemaeventoModel	7	Baja	Baja	Alta
DatBuzonNotif	3	Baja	Baja	Alta
DatMessagebus	4	Baja	Baja	Alta

Tabla 5 Evaluación de la métrica (TOC).

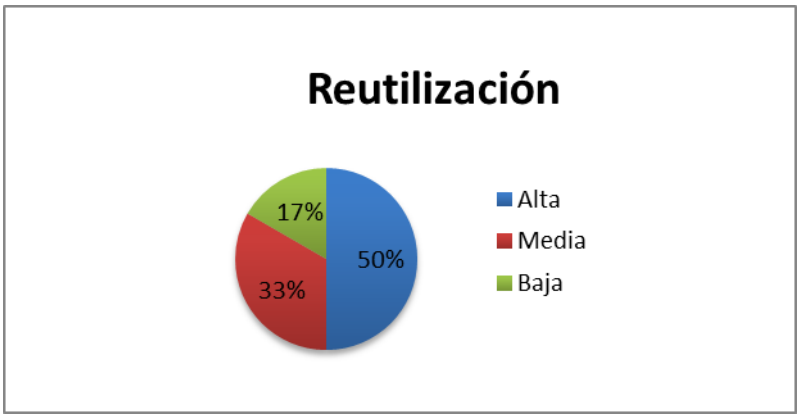
Los elementos: **Gráfica 1, Gráfica 2, Gráfica 3, Gráfica 4 y Gráfica 5** guardan relación con la **Tabla5**, y representan gráficamente el resultado de haber aplicado (TOC).



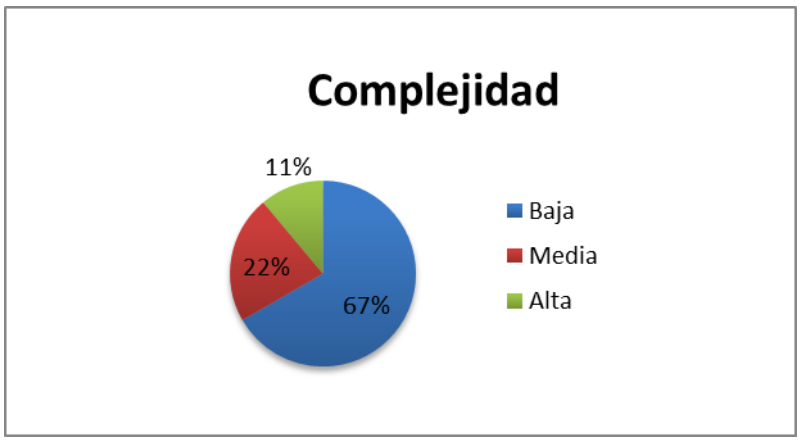
Gráfica 1 Cantidad de procedimientos (TOC).



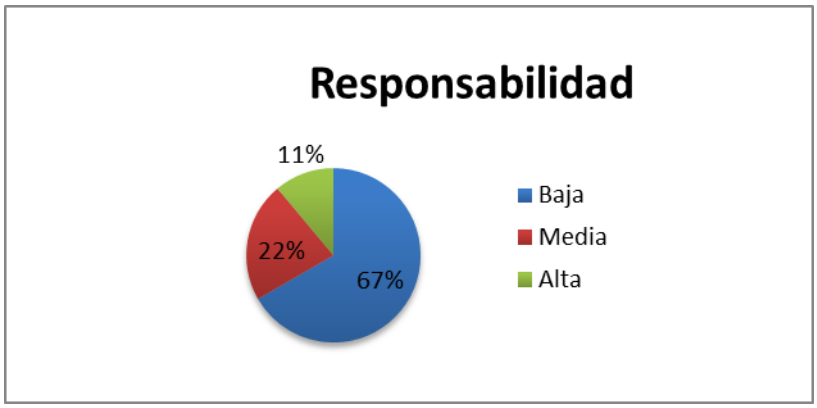
Gráfica 2 Representación de las clases según la cantidad de operaciones.



Gráfica 3 Resultados obtenidos de la evaluación de la métrica TOC para el atributo **Reutilización**.



Gráfica 4 Resultados obtenidos de la evaluación de la métrica TOC para el atributo **Complejidad**.



Gráfica 5 Resultados obtenidos de la evaluación de la métrica TOC para el atributo **Responsabilidad**.

3.3.3 Análisis de los resultados obtenidos al evaluar la métrica TOC

Haciendo un análisis de los resultados obtenidos para los atributos de la métrica TOC apreciables en las gráficas de pastel mostradas anteriormente, se puede observar que la mayoría de las clases que conforman el sistema para los atributos responsabilidad y complejidad están dentro de la categoría media y Baja para un 55% y 36% respectivamente del total, lo que representa que las

clases no tienen mucha responsabilidad y no son tan complejas, y en caso de ocurrir algún cambio en el sistema de clases del componente, estaría menos comprometido el funcionamiento correcto del mismo. Mientras que el atributo reutilización cuenta con igual porciento pero en las categorías alta y media mostrando así que el componente cuenta con una elevada reutilización que permite que las clases puedan ser utilizadas e instanciadas por otras. Por lo que se concluye que los resultados obtenidos según esta métrica son positivos.

3.3.4 Relaciones entre clases (RC)

Está dado por el número de relaciones de uso de una clase con otra y evalúa los atributos de calidad que se muestran en la **Tabla 6**.

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.

Tabla 7 Atributos de calidad que mide (RC).

Para los cuales están definidos los siguientes criterios y categorías de evaluación:

Atributo	Categoría	Criterio
Acoplamiento	Ninguna	0
	Baja	1
	Media	2
	Alta	>2
Complejidad del mantenimiento	Baja	<=Promedio
	Media	Entre Promedio y 2 * Promedio

	Alta	>2*Promedio
Reutilización	Baja	>2*Promedio
	Media	Entre Promedio y 2 * Promedio
	Alta	<=Promedio
Cantidad de pruebas	Baja	<=Promedio
	Media	Entre Promedio y 2 * Promedio
	Alta	>2*Promedio

Tabla 8 Criterios y categorías para evaluar la métrica (RC).

3.3.5 Resultados obtenidos al aplicar (RC)

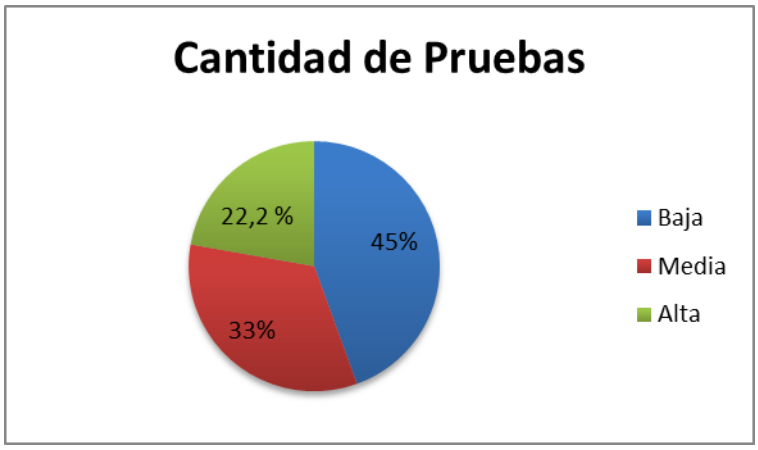
De 9 clases existentes constan de un total de **7** dependencias entre ellas, a continuación en la siguiente tabla estará reflejada junto a los atributos de calidad de Acoplamiento, Complejidad de Mantenimiento, Reutilización y Cantidad de Pruebas. Se obtuvo un promedio de relaciones de dependencia por clase de **0.77**.

Clase	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad Mant.	Reutilización	Cantidad de Pruebas
AplicacionController	1	Baja	Media	Media	Media
GestionarController	2	Media	Alta	Baja	Alta
GestnotificacionesController	1	Baja	Media	Media	Media
DatBuzonNotifModel	1	Baja	Media	Media	Media
DatMessagebusModel	2	Alta	Alta	Baja	Alta
GestNotificacionesModel	0	Ninguno	Baja	Alta	Baja
SubsistemaeventoModel	0	Ninguno	Baja	Alta	Baja
DatBuzonNotif	0	Ninguno	Baja	Alta	Baja

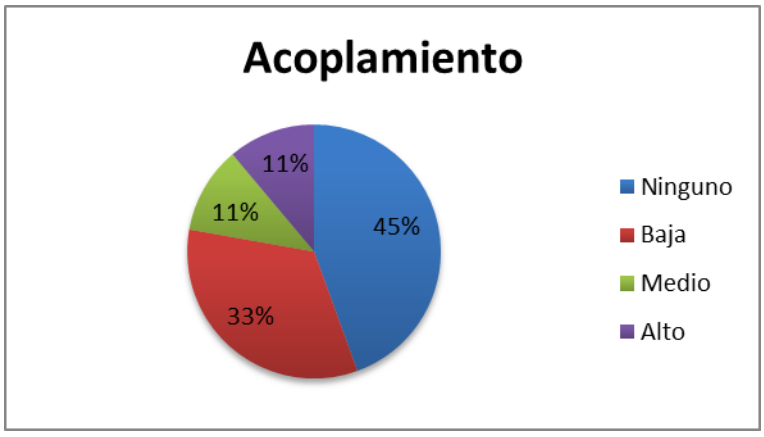
DatMessagebus	0	Ninguno	Baja	Alta	Baja

Tabla 9 Evaluación de la métrica (RC).

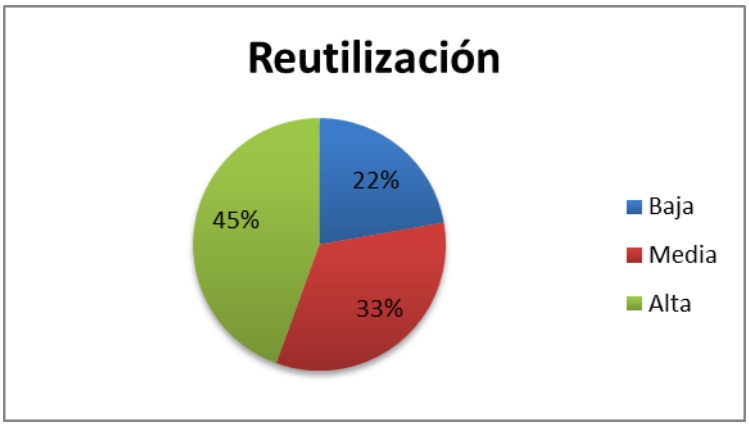
Los elementos: **Gráfica 6**, **Gráfica 7**, **Gráfica 8**, **Gráfica 9** guardan relación con la **Tabla 9**, y representan gráficamente el resultado de haber aplicado (RC).



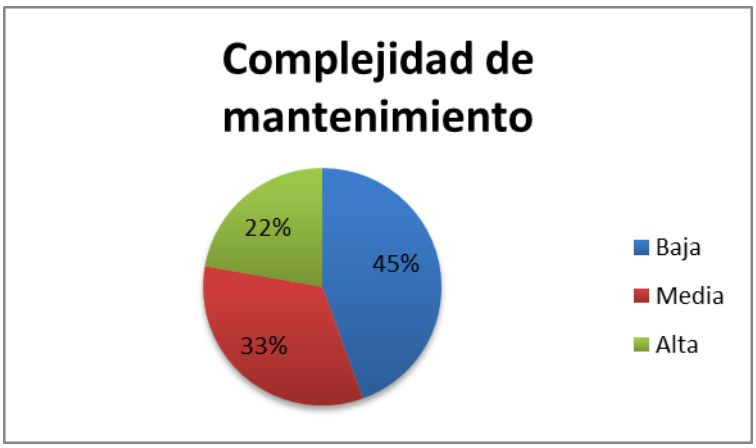
Gráfica 6 Resultados obtenidos de la evaluación de la métrica RC para el atributo **Cantidad de pruebas**.



Gráfica 7 Resultados obtenidos de la evaluación de la métrica RC para el atributo **Acoplamiento**.



Gráfica 8 Resultados obtenidos de la evaluación de la métrica RC para el atributo **Reutilización**.



Gráfica 9 Resultados obtenidos de la evaluación de la métrica RC para el atributo **Mantenimiento**.

3.3.6 Análisis de los resultados obtenidos al evaluar la métrica RC

Los resultados obtenidos durante la evaluación del instrumento de medición de la métrica RC demuestran que las clases del diseño poseen un bajo acoplamiento, ya que para este atributo las categorías Ninguno y Alto sumaron un 82% del total, mostrando igual porciento en la categoría alta y media del atributo reutilización. Los atributos complejidad de mantenimiento y cantidad de pruebas, sumaron un 82 % en las categorías baja y media, lo que demuestra que no es necesario un elevado esfuerzo en el momento de realizar cambios, rectificaciones y pruebas al software.

3.4 Pruebas de software

Las pruebas de software consisten en la dinámica de la verificación del comportamiento de un programa en un conjunto finito de casos de prueba debidamente seleccionados, por lo general en infinitas ejecuciones de dominio, contra el comportamiento esperado. Son una serie de actividades que se realizan con el propósito de encontrar los posibles fallos de implementación, calidad o usabilidad de un programa u ordenador; probando el comportamiento del mismo (Pressman, 2005).

La prueba de software es un elemento crítico para la garantía del correcto funcionamiento del software. Entre sus objetivos están:

1. Detectar defectos en el software.
2. Verificar la integración adecuada de los componentes.
3. Verificar que todos los requisitos se han implementado correctamente.
4. Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
5. Diseñar casos de prueba que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo (Pressman, 2005).

Dentro de los tipos de prueba que existen se encuentran las pruebas de caja blanca y caja negra. Ambas están definidas por CEIGE para realizar las pruebas de software a las aplicaciones web que se desarrollen en el centro. A continuación se exponen algunas características de ambos tipos de pruebas:

3.4.1 Pruebas de caja blanca

La prueba de caja blanca se basa en el diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivarlos. Mediante la prueba de la caja blanca el ingeniero del software puede obtener casos de prueba que:

1. Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
2. Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
3. Ejecuten todos los bucles en sus límites operacionales.
4. Ejerciten las estructuras internas de datos para asegurar su validez.

Es por ello que se considera a la prueba de Caja Blanca como uno de los tipos de pruebas más importantes que se le aplican a los software, logrando que disminuya en un gran porciento, el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad (Pressman, 2005).

3.4.1.1 Camino Básico

La prueba del camino básico es una técnica de prueba de la Caja Blanca propuesta por Tom McCabe. Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática. Los pasos que se siguen para aplicar esta técnica son (Pressman, 2005):

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo.
3. Se determina un conjunto básico de caminos independientes.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Luego de entender todos los pasos, se realizan las pruebas de caja blanca, mostradas a continuación mediante la técnica del camino mínimo del método: **ModificarNotificación** de la clase **GestNotificacionesModel**. Este método tiene la responsabilidad de modificar una notificación, para lo cual modifica directamente sobre el fichero “subemitennotif.xml”. Recibe 3 variables como parámetro: el código de la notificación a modificar que es un atributo único que identifica inequívocamente una notificación, el mensaje literal correspondiente a dicho código, formando los dos de conjunto una notificación y el sistema el que pertenece dicha notificación el cual contiene un conjunto de notificaciones. En la Figura 8 se muestra el código fuente de dicho método.

```php
public function ModificarNotificacion($sistema, $codigo, $notificacion) {
    $xml = ZendExt_FastResponse::getXML('subsemitennotif');
    $register = Zend_Registry::getInstance();
    $subsemitennotif = $register->config->xml->subsemitennotif;
    $sis = strtolower($sistema);
    $sist = $this->limpia_espacios($sis);
    $notificaciones = $xml->$sist->children();
    $cantidad = count($notificaciones);
    if (!$this->EstaAsociada($sistema, $codigo)) {
        for ($i = 0; $i < $cantidad; $i++) {
            if ($notificaciones[$i]['cod'] == $codigo) {
                unset($notificaciones[$i]);
            }
        }
        $this->ActualisarMsgBus($sist, $codigo, $notificacion, "mod");
        $nuevanotificacion = $xml->$sist->addChild("" . 'notificacion');
        $nuevanotificacion->addAttribute('cod', $codigo);
        $nuevanotificacion->addAttribute('literal', $notificacion);
        file_put_contents($subsemitennotif, $xml->asXml());
        $this->catalogo->Actualizar();
        echo json_encode(array('mensaje' => 'Se modificó la notificación satisfactoriamente.', 'CoDMsg' => 1));
    } else {
        echo json_encode(array('mensaje' => 'Notificación asociada a una sistema. No se puede modificar.', 'CoDMsg' => 1));
    }
}
```

Figura 14 Función escogida para aplicar la prueba de caja blanca.

Paso1 A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.

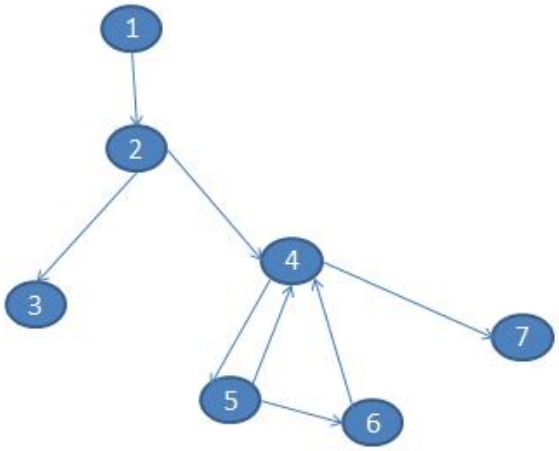


Figura 15 Grafo de flujo asociado.

Paso 2 Se calcula la complejidad Ciclomática.

Existen 3 vías para calcular la complejidad ciclomática, V (G), de un grafo de flujo asociado:

1. La cantidad de nodos predicado que existan en el grafo.
2. V (G) = A - N + 2.

Donde: *A* es el número de aristas del grafo y *N* es el número de nodos.

3. La complejidad ciclomática, V(G), también se define como:
   V (G) = P + 1.
   Donde: *P* es el número de nodos predicado contenido en el grafo G.

Se procede a calcular la complejidad ciclomática del grafo mediante la segunda vía V (G) = A - N + 2. Tras evaluar en la función queda de la siguiente forma:

V (G) = 8 – 7 + 2

Lo cual da como resultado 3, por tanto la complejidad ciclomática del grafo de la **Figura 16** es 3, que coincide con la cantidad de caminos independientes, el límite superior para la cantidad de casos de prueba que se pueden realizar para forzar la ejecución de la función, por cada uno de los caminos independientes que contiene el grafo.

Paso 3 Determinar un conjunto básico de caminos independientes.

Un camino independiente se debe mover por lo menos por una arista que no haya sido recorrida anteriormente. Por lo tanto los caminos independientes para este caso son:

Camino básico Nº 1: 1-2-3

Camino básico Nº 2: 1-2-4-5-6-7

Camino básico Nº 3: 1-2-4-7

Paso 4 Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Caso de prueba para camino básico Nº 1 (1-2-3)

Si $this->EstaAsociada =true entonces mostrar mensaje (“Notificación asociada a una sistema. No se puede modificar.”)

Caso de prueba para camino básico Nº 2 (1-2-4-5-6-7)

Si $this->EstaAsociada =true, buscar si la notificación que se quiere modificar existe y si existe entonces eliminarla, luego crear una notificación con los datos modificados y luego mostrar mensaje (“Se modificó la notificación satisfactoriamente.”)

Caso de prueba para camino básico Nº 2 (1-2-4-5-6-7)

Si $this->EstaAsociada =true, buscar si la notificación que se quiere modificar existe y si no existe entonces crear una notificación con los datos modificados y luego mostrar mensaje (“Se modificó la notificación satisfactoriamente.”)

De esta forma se culmina la prueba de caja blanca con el complimiento de los tres caminos independientes y se llega a la conclusión que la prueba resultó satisfactoria para los casos de prueba y se demuestra que al emplear este tipo de pruebas se tiende a que disminuya en un gran porciento el número de errores existentes en los sistemas, obteniendo mayor calidad y confiabilidad del código fuente y por lo tanto de la aplicación.

3.4.2 Prueba de caja negra

Las pruebas de caja negra fueron aplicadas a las interfaces de usuarios por el grupo de trabajo del Dpto. de Tecnología. Fueron necesarias 2 iteraciones ya que durante la primera iteración se hallaron 22 no conformidades y en la segunda iteración no se encontró ninguna, por lo que el componente para la notificación de notificaciones estaba libre de no conformidades culminando así la fase de pruebas internas, donde consta que el producto puede ser liberado.

La tabla que se muestra a continuación ofrece detalles de los resultados obtenidos hasta el momento, desglosando las no conformidades (NC) en significativas (S) que incluyen errores de validación, de función y excepciones, las que no proceden (NP) y las no significativas NS que incluyen los errores en los casos de prueba.

Agrupación de Requisitos	**No conformidades**							
	Iteración 1				Iteración 2			
	NC	NP	S	NS	NC	NP	S	NS
Gestionar notificaciones	7		0	7	0	0	0	0

Tabla 10 Resultados de la prueba.

En las mismas se encontraron mayormente errores de faltas de ortografías en las interfaces, de validaciones, los mensajes de información estaban mal elaborados, faltaban los íconos de los botones, etc.

Luego de ser corregidos los errores encontrados en las pruebas, se pudo comprobar que el flujo de trabajo de las interfaces estaban correcto ya que cumplían con las condiciones necesarias que se habían planteado.

3.5 Conclusiones parciales del capítulo

Se implementó el componente de gestión de notificaciones basado en reglas de negocio para el marco de trabajo Sauxe logrando notificar a los usuarios, de eventos que ocurren dentro del marco de trabajo en tiempo real. Se validó la solución desarrollada empleando técnicas para la validación de los requisitos, métricas para la validación del diseño y pruebas de caja blanca y caja negra para la validación de la aplicación. Todo esto permitió garantizar la obtención de un software de alta calidad.

CONCLUSIONES GENERALES

Con el desarrollo del presente trabajo de diploma se culminó el objetivo principal de la investigación; desarrollar un componente de gestión de notificaciones para el marco de trabajo sauxe basado en reglas de negocio. Para obtener dichos resultado, se cumplieron los siguientes aspectos:

- Se construyó el marco teórico de la investigación, en el que realizó un estudio de los sistemas similares con el objetivo de reutilizarlos o estudiarlos para adquirir experiencias y comprobar si cumplían las expectativas del centro.

- Se realizó el análisis y diseño de la solución, puesto que al concluir el marco teórico se llegó a la conclusión que no existían sistemas que cumplieran las expectativas del centro y se necesitaba el desarrollo de uno que abarcara las funcionalidades y objetivos del componente para incrementar el nivel de acceso a la información por el usuario en el marco de trabajo sauxe. Este aspecto incluyó la generación de los artefactos que propone el modelo de desarrollo utilizado.

- Se realizó la implementación y prueba de la solución, donde se cumplimentó el flujo para el desarrollo del componente. Se validó el diseño propuesto y la implementación, ambos con resultados satisfactorios según las pruebas escogidas y realizadas.

Tras culminados los tres capítulos con los que cuenta la presente investigación se puede afirmar que:

- Las métricas orientadas a las clases empleadas, validaron que se realizó un diseño correcto de la solución.

- Las pruebas de caja blanca y funcionales realizadas a la aplicación, validaron el correcto funcionamiento de la misma.

Trabajos citados

Baryolo, Dr y Gómez, Oiner. 2010. *Solución informática de autorización en entornos multientidad y multisistema.* Universidad de las Ciencias Informáticas : s.n., 2010.

Bock, Heiko. 2012. *Tha definitive guide ti netbeans plataform 7.* 2012.

Business Rules Group. [En línea] http://www.businessrulesgroup.org/home-brg.shtml.

CEIGE, Centro de Informatización de Entidades. 2013. *Modelo de Desarrollo de Software.* Habana : Subdirección de Producción, 2013. 1.2.

Chisholm, M. 2013. *How to Build a Business Rules Engine: Extending Application Functionality through Metadata Engineering.* San Francisco : Morgan Kaufmann Publishers Inc., 2013.

Club-BPM. 2009. *BPM Business Process Management - Gestión de Procesos de Negocios.* 2009.

—. 2009. *BPM Business Process Management - Gestión de Procesos de Negocios.* 2009.

Community, Ignite Realtime. 2011. *Ignite realtime. Ignite realtime.* [En línea] 2011. [Citado el: 2012 de 11 de 10.] http://www.igniterealtime.org/.

Conceptos de procesos. 2012. Conceptos de procesos. [En línea] 15 de Mayo de 2012. http://es.scribd.com/doc/49282475/Conceptos-Procesos.

Corporation, Rational Software. 2003. *Glosario de Rational Unified Process.* 2003.

Díaz, Estanque y otros. 2013. *Diseño del mecanismo para alertas y notificaciones del sistema de información hospitalaria ALAS HIS.* Habana : s.n., 2013.

Diaz, Javier. 2012. *PHP: una solución "open source" para el desarrollo de páginas Web dinámicas.* 2012.

Doctrine. 2007. Doctrine. Doctrine. [Online] Doctrine. [En línea] 2007. http://www.doctrine-project.org/.

DRAKE, Joshua y WORSLEY. 2011. *Practical PostgreSQL. s.l.* O'Reilly Media : s.n., 2011.

Ecured. 2011. Ecured. [En línea] 12 de 12 de 2011. http://www.ecured.cu/index.php/Modelo_de_dominio.

Eguíluz Pérez, Javier. 2009. *Introducción a JavaScript.* 2009.

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

Frederick, S y otros. 2009. *Learning EXT JS.* 2009.

Garcia, Jr Jesus D. 2010. *ExtJs in action.* 2010.

GARIMELLA, KIRAN, LEES, MICHAEL y WILLIAM, BRUCE. 2012. *BPM (GERENCIA DE PROCESOS DE NEGOCIO).* 2012.

Gartner Technology Business Research Insight. [En línea] http://www.gartner.com/.

González, Oscar Oramas. 2011. *Interoperabilidad del proceso inventario en el sistema Cedrux.* 2011.

Gruber, T. 2007. Ontología - Encyclopedia of Database Systems.. [En línea] 2007. http://tomgruber.org/writing/ontology-definition-2007.htm.

Hay y Hayle. 1997. *GUIDE Business Rules Project.* Chicago : Tech. rep, 1997.

Hay, D y Hayle, K A. *Defining Busines.*

Hevia, Andrés. 2013. Pensando en SOA.com. [En línea] 14 de 04 de 2013. http://pensandoensoa.com/category/5-procesos-y-reglas/.

IEEE. 2010. *Introducción a la Ingeniería de Requisitos. Ingeniería de software.* 2010.

2013. Informática . [En línea] 2013. http://www.informatica2013.sld.cu/index.php/informaticasalud/2013/paper/viewFile/160/174.

Landívar, Edgar. 2009. *Comunicaciones Unificadas con Elastix. s.l.* Segunda Edición . 2009.

Larman, Craig. 1999. *Applying UML & Patterns: Introduction to Object-Oriented Analysis & Design, & Iterative Development.* 1999.

M, S Perez. 2009. *Propuesta de modelo de desarrollo de software tecnológico del Centro de Soluciones de Gestión.* Cuidad de la Habana : s.n : s.n., 2009.

María José Escalona y Nora Koch. 2002. *Ingeniería de Requisitos en Aplicaciones para la Web.* Sevilla : Un estudio comparativo, 2002.

Moffitt, Jack. 2010. *XMPP Programming with Javascript® and jQuery.* s.l. : Indianapolis : s.n, 2010.

Mohammed J, Kabir. 2004. *La biblia server apache.* 2004.

Momjian, Bruce. 2001. *PostgreSQL Introduction and Concepts.* 2001.

Montes de Oca, Martínez del Busto y Gonzalez. 2008. *Modelación orientada a hechos en el enfoque de reglas de negocio. Tech. rep.* s.l. : Informática en Salud, 2008.

Mozilla. 2012. Mozilla Firefox. Mozilla Firefox. [En línea] Mozilla. [En línea] 2012. [Citado el: 2013 de 11 de 15.] http://www.mozilla.org/es-ES/firefox/features/.

Muñoz, Fernandez. 2007. Procesamiento del lenguaje natural para recuperación de información. [En línea] 2007. http://procesamientolenguajerecuperacion.50webs.org/.

Pérez Alfonso, Damian. 2012. *Normas y estándares de codificación.* Universidad de las Ciencias Informáticas : s.n., 2012.

Pérez, M S. 2009. *Propuesta de modelo de desarrollo de software tecnológico del Centro de Soluciones de Gestión.* Ciudad de la Habana : s.n., 2009.

Petter Saint y otros. 2009. *XMPP: The Definitive Guide,Building Real-Time Applications with Jabber.* 1ra. s.l. : Sebastopol, 2009.

Plus, ManageEngine ServiceDesk. 2009. [En línea] 2009. http://www.manageengine.com/products/service-desk/spanish/.

—. 2009. ServiceDesk Plus Software Help Desk alineado a ITIL con gestión de archivos. [En línea] 2009. http://www.manageengine.com.mx/service-desk/business-rules.html?ftr.

Pressman, Roger S. 2002. *Ingeniería del software.* 2002.

—. 2005. *Ingienería del software, un enfoque práctico. s.l.* 5ta. 2005.

Rapidsvn. 2013. Rapidsvn. Rapidsvn. [En línea] Rapidsvn. [En línea] 2013. [Citado el: 25 de 3 de 2013.] http://www.rapidsvn.org/.

Roberto y otros. 2006. *Metodología de la Investigación Científica. Iztapalapa,.* México : s.n. : 4ta Edición, 2006.

Ross, R G. 2005. *Business Rule Concepts - Getting to the Point of Knowledge.* s.l. : Business Rule Solutions Inc, 2005.

Ross, R. G. 2010. *The Business Rule Book: Classifying, Defining and Modeling.* Universidad Estatal de Pensilvania : 2, ilustrada, 2010.

Sommerville, I. 2002. *Ingeniería de Software, Pearson Educación.* 2002.

Sommerville, Ian y Sawyer, Pete. 2005. *Requirements Engineering: A good practice guide.* 2005.

Tedeschi, Nicolás. 2012. Microsoft MSDN. ¿Qué es un Patrón de Diseño? [En línea] 13 de marzo de 2012. http://msdn.microsoft.com/es-es/library/bb972240.aspx.

Vaswani, Vikram. 2010. *Zend Framework: A Beginner's Guide.* 2010.

Villa, Universidad "Martha Abreu" de "Las. 2011. Medical application for kidney transplantation using business rules. [En línea] 2011. http://bvs.sld.cu/revistas/rhab/v11n1/rhcm21112.htm.

Visual-Paradigm. 2013. Visual-Paradigm. Visual-Paradigm. [En línea] Visual-Paradigm. [En línea] 2013. [Citado el: 2013 de 3 de 5.] http://www.visual-paradigm.com/.

Werdmuller, Ben. 2010. *Build a web-based notification tool with XMPP Write real-time web applications with XMPP, PHP, and JavaScript.* 2010.

- Que se muestre las notificaciones en forma de teletipos para una mejor visualización de estas.

Anexe 1. Acta deliberación

Anexo 2. Especificación de requisitos funcionales.

Modulo Gestionar notificaciones

Tabla 11 Descripción del requisitos funcional Adicionar notificación que emiten los subsistemas.

Precondiciones:	El usuario debe estar autenticado en el sistema y debe poseer permisos para ejecutar esta acción.
Descripción	
Adiciona a un subsistema la notificación (código y mensaje) que emite.	
Flujo de eventos	
Flujo básico de eventos	
1 El usuario Selecciona la interfaz Gestionar notificaciones.	
2 Seleccionar un sistema.	
3 Seleccionar la opción Adicionar en la interfaz.	
4 Introduce los datos.	
5 Escoger la opción Aceptar.	
Post-Condiciones	
1 El sistema muestra un mensaje: ¨Se adicionó la notificación satisfactoriamente¨.	
Validaciones	
Se validan los datos según lo establecido en el Modelo conceptual.	
Conceptos	
Visible en la interfaz: sistema, código y mensaje.	

Tabla 12 Descripción del requisitos funcional Eliminar notificación que emiten los subsistemas.

Precondiciones	El usuario debe estar autenticado en el sistema y debe poseer permisos para ejecutar esta acción.
Descripción	
Elimina a un subsistema la notificación (código y mensaje) que emite.	
Flujo de eventos	
Flujo básico de eventos	
1 El usuario Selecciona la interfaz Gestionar notificaciones.	
2 Seleccionar un sistema.	
3 Seleccionar una notificación.	

4 Seleccionar la opción Eliminar en la interfaz.
5 El sistema muestra un mensaje: ¨¿Está seguro de eliminar esta notificación? Se eliminarán todas las notificaciones que tengan que ver con ella¨.
6 Escoger la opción Aceptar.
Post-Condiciones
1 El sistema muestra un mensaje: ¨Se limpió el usuario satisfactoriamente¨.
Flujos alternativos 6.a Notificación que ya está asociada a una sistema.
1 El sistema muestra un mensaje: ¨Notificación asociada a un sistema. No se puede eliminar.¨.
2 Volver al paso 3 del flujo básico.
Validaciones
Se validan los datos según lo establecido en el Modelo conceptual
Conceptos
Visible en la interfaz: sistema, notificación.
Elimina una notificación (código y mensaje) de un sistema.

Tabla 13 Descripción del requisitos funcional Modificar notificación que emiten los subsistemas.

Precondiciones:	El usuario debe estar autenticado en el sistema y debe poseer permisos para ejecutar esta acción.
Descripción	
Modifica a un subsistema la notificación (código y mensaje) que emite.	
Flujo de eventos	
Flujo básico de eventos	
1 El usuario Selecciona la interfaz Gestionar notificaciones.	
2 Seleccionar un sistema.	
3 Seleccionar la notificación que desea modificar.	
4 Seleccionar la opción Modificar en la interfaz.	
5 Introduce los datos.	
6 Seleccionar la opción Aceptar.	
Post-Condiciones	
1 El sistema muestra un mensaje: ¨Se modificó la notificación Satisfactoriamente¨.	
Validaciones	
Se validan los datos según lo establecido en el Modelo conceptual.	
Conceptos	

Visible en la interfaz: sistema, código y mensaje.

Tabla 14 Descripción del requisitos funcional Adicionar notificación que reciben los subsistemas.

Precondiciones:	El usuario debe estar autenticado en el sistema y debe poseer permisos para ejecutar esta acción.
Descripción	
Adiciona una notificación (código y mensaje) al subsistema que recibe.	
Flujo de eventos	
Flujo básico de eventos	
1 El usuario Selecciona la interfaz Gestionar notificaciones.	
2 Seleccionar un sistema.	
3 Seleccionar una notificación.	
4 Marcar el servicio del sistema que a través del cual se recibirá la notificación.	
5 Escoger la opción Guardar cambios.	
Post-Condiciones	
1 El sistema muestra un mensaje: ¨Se realizaron los cambios satisfactoriamente.¨.	
Validaciones	
Se validan los datos según lo establecido en el Modelo conceptual.	
Conceptos	
Visible en la interfaz: sistema, notificación (código y mensaje), sistema a recibir, servicio.	

Tabla 15 Descripción del requisitos funcional Eliminar notificación que reciben los subsistemas.

Precondiciones	El usuario debe estar autenticado en el sistema y debe poseer permisos para ejecutar esta acción.
Descripción	
Elimina una notificación (código y mensaje) al subsistema que recibe.	
Flujo de eventos	
Flujo básico de eventos	
1 El usuario Selecciona la interfaz Gestionar notificaciones.	
2 Seleccionar un sistema.	
3 Seleccionar una notificación.	
4 Desmarcar el servicio del sistema que a través del cual se recibirá la notificación.	
5 Escoger la opción Guardar cambios.	

Post-Condiciones
1 El sistema muestra un mensaje: ¨Se realizaron los cambios satisfactoriamente.¨.
Validaciones
Se validan los datos según lo establecido en el Modelo conceptual.
Conceptos
Visible en la interfaz: sistema, notificación (código y mensaje), sistema a recibir, servicio.

Tabla 16 Descripción del requisitos funcional Modificar notificación que reciben los subsistemas.

Precondiciones:	El usuario debe estar autenticado en el sistema y debe poseer permisos para ejecutar esta acción.
Descripción	
Modificar una notificación (código y mensaje) al subsistema que recibe.	
Flujo de eventos	
Flujo básico de eventos	
1 El usuario Selecciona la interfaz Gestionar notificaciones.	
2 Seleccionar un sistema.	
3 Seleccionar una notificación.	
4 Marcar otro servicio del sistema que a través del cual se recibirá la notificación.	
5 Escoger la opción Guardar cambios.	
Post-Condiciones	
1 El sistema muestra un mensaje: ¨Se realizaron los cambios satisfactoriamente.¨.	
Validaciones	
Se validan los datos según lo establecido en el Modelo conceptual.	
Conceptos	
Visible en la interfaz: sistema, notificación (código y mensaje), sistema a recibir, servicio.	

Tabla 17 Descripción del requisitos funcional Listar notificaciones que emiten los subsistemas.

Precondiciones:	El usuario debe estar autenticado en el sistema y debe poseer permisos para ejecutar esta acción.
Descripción	
Adiciona una notificación (código y mensaje) al subsistema que emite.	
Flujo de eventos	
Flujo básico de eventos	

1 El usuario Selecciona la interfaz Gestionar notificaciones.
2 Seleccionar un sistema.
Validaciones
Se validan los datos según lo establecido en el Modelo conceptual.
Conceptos
Visible en la interfaz: sistema, código y mensaje.

Tabla 18 Descripción del requisitos funcional Listar servicios.

Precondiciones:	El usuario debe estar autenticado en el sistema y debe poseer permisos para ejecutar esta acción.
Descripción	
Listar servicios de un subsistema que recibe notificaciones.	
Flujo de eventos	
Flujo básico de eventos	
1 El usuario Selecciona la interfaz Gestionar notificaciones.	
2 Seleccionar un sistema.	
3 Seleccionar una notificación.	
4 Seleccionar un subsistema que recibe.	
Validaciones	
Se validan los datos según lo establecido en el Modelo conceptual.	
Conceptos	
Visible en la interfaz: sistema, notificación (código y mensaje), sistema a recibir, servicio.	

Tabla 19 Descripción del requisitos funcional Listar subsistemas que emiten.

Precondiciones:	El usuario debe estar autenticado en el sistema y debe poseer permisos para ejecutar esta acción.
Descripción	
Listar los subsistemas registrados que emiten notificaciones.	
Flujo de eventos	
Flujo básico de eventos	
1 El usuario Selecciona la interfaz Gestionar notificaciones.	
Validaciones	
Se validan los datos según lo establecido en el Modelo conceptual.	
Conceptos	
Visible en la interfaz: sistema, código y mensaje.	

Anexo 3. Diagramas de secuencia.

Figura 17 Diagrama de secuencia Adicionar notificación recibida.

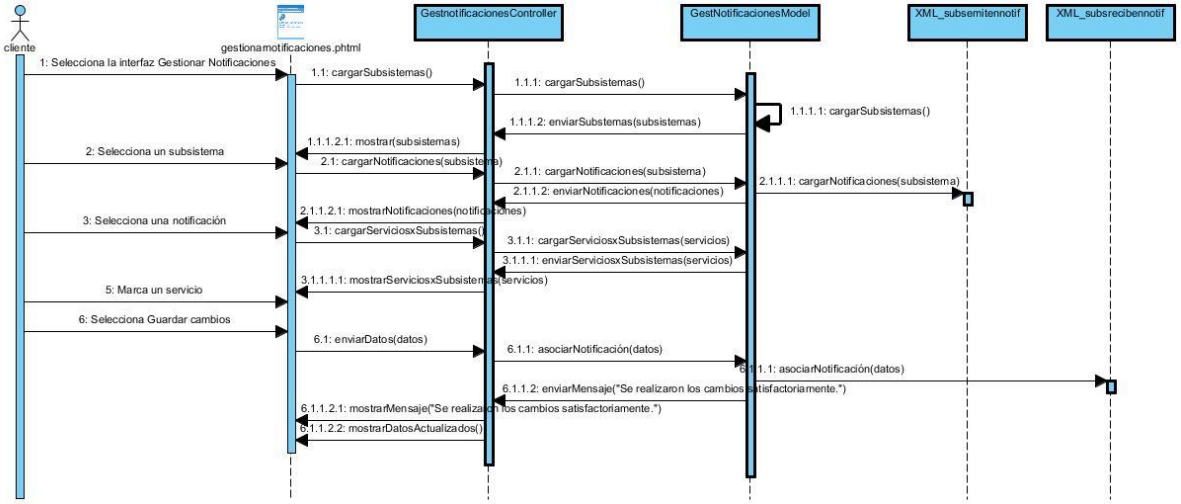


Figura 18 Diagrama de secuencia Eliminar notificación emitida.

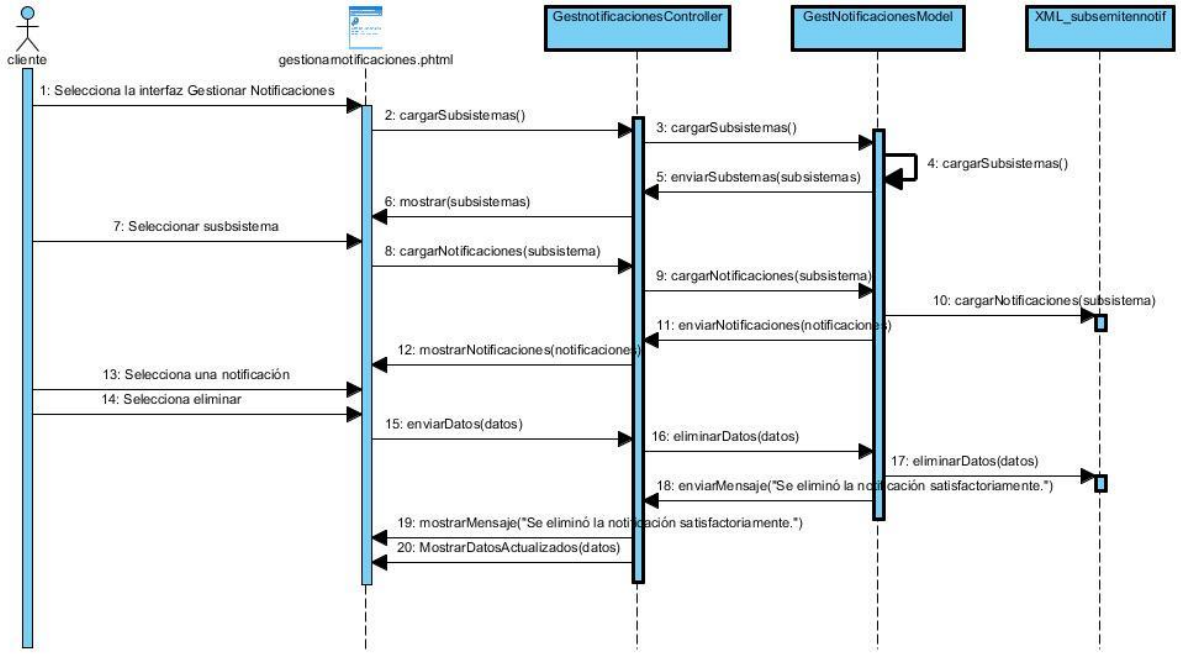


Figura 19 Diagrama de secuencia Eliminar notificación recibida.

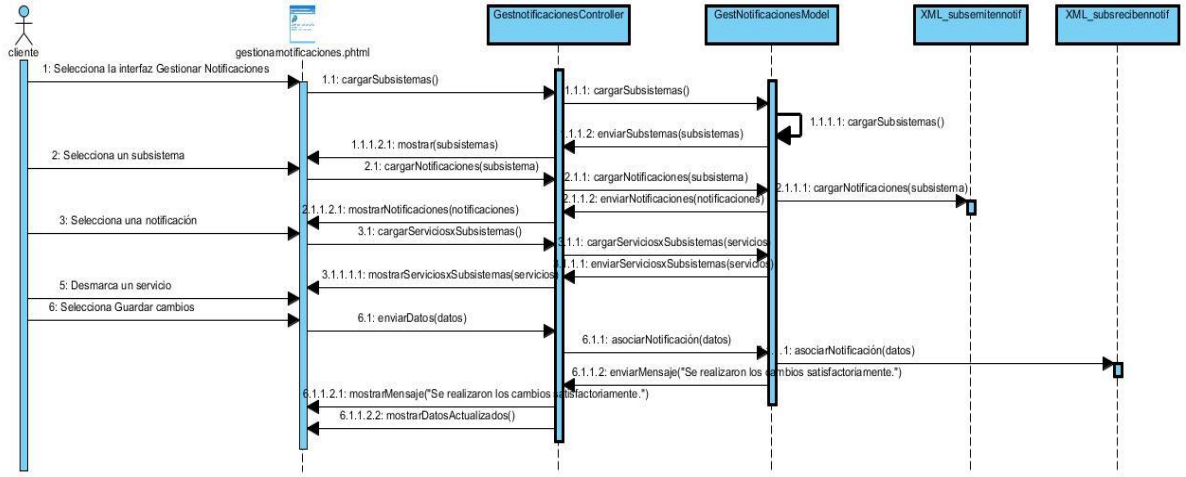


Figura 20 Diagrama de secuencia Listar notificación.

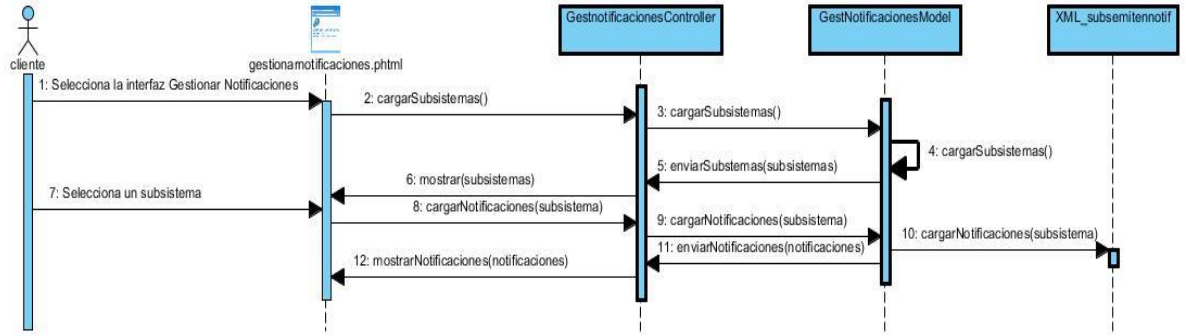


Figura 21 Diagrama de secuencia Listar servicio.

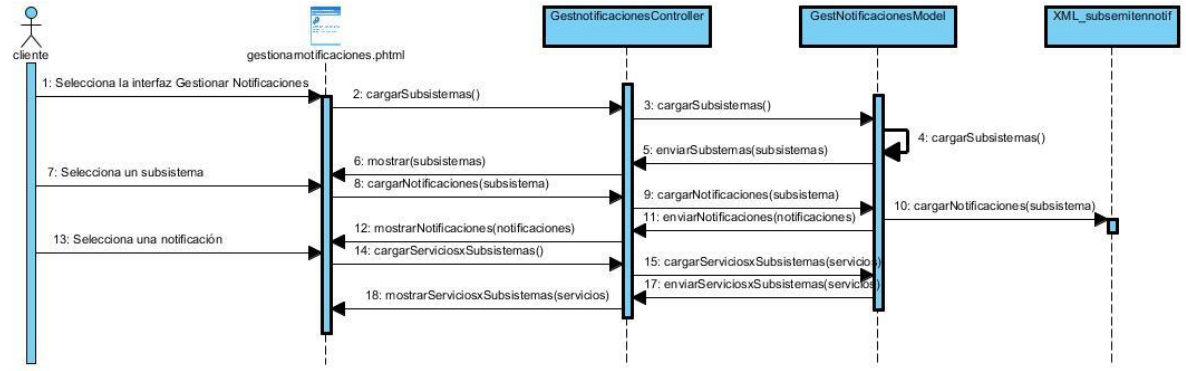


Figura 22 Diagrama de secuencia Listar subsistema.

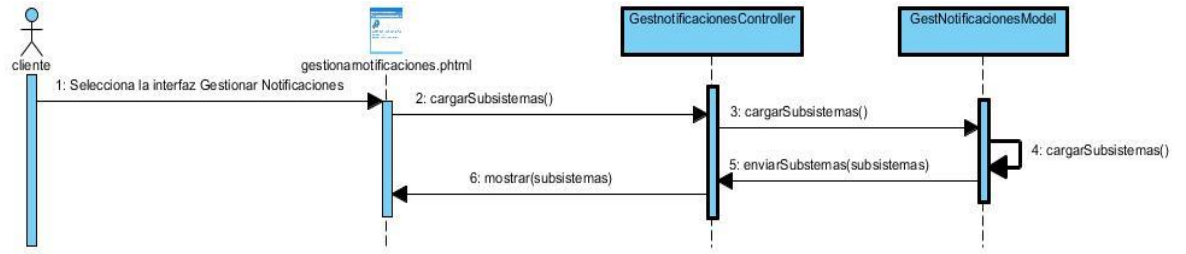


Figura 23 Diagrama de secuencia Modificar notificación emitida.

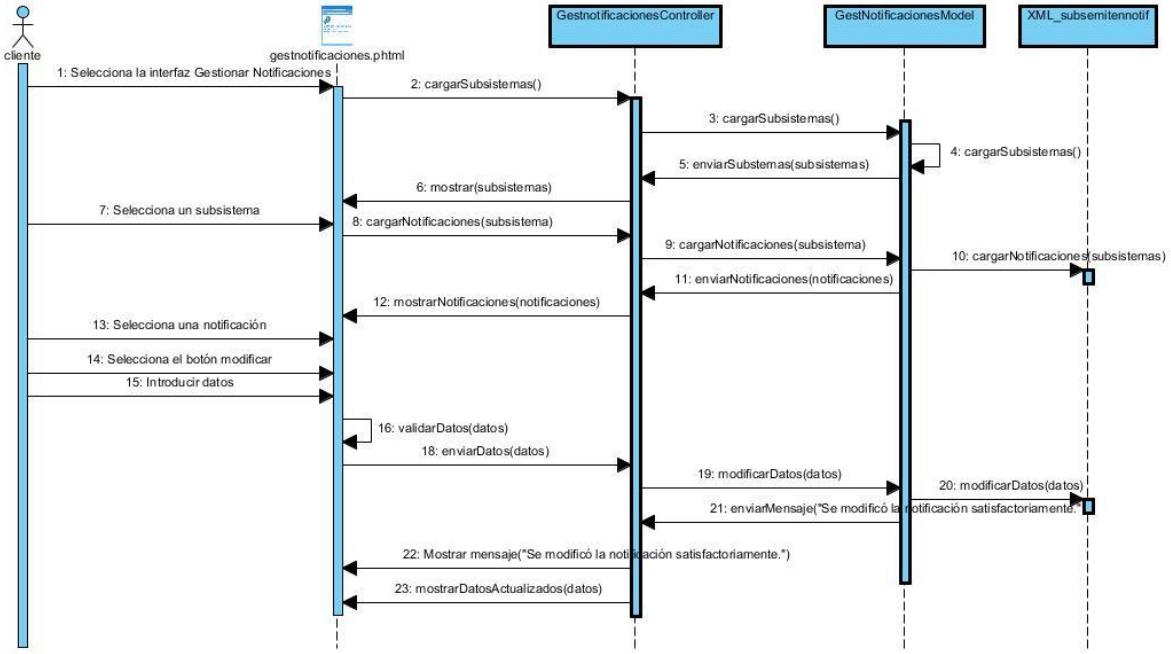


Figura 24 Diagrama de secuencia Modificar notificación recibida.