Universidad de las Ciencias Informáticas

Facultad 3

Título: Extensión del componente de gestión de notificaciones del marco de trabajo Sauxe para el envío de alertas a nivel de roles y usuarios mediante reglas de negocio.

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

Autor:

Ernesto Cuendias Pérez

Tutores:

Msc. Orlando Arnaldo Valenzuela Aguilera

Ing. Pedro Manuel Alás Verdecia

La Habana, 2015

"Año 57 de la Revolución"

"No podéis conectar los puntos mirando hacia el futuro; solo podéis conectarlos mirando hacia el pasado. Por lo tanto, tenéis que confiar en que los puntos, de alguna manera, se conectarán en vuestro futuro."

Steve Jobs

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 _____.

Ernesto Cuendias Pérez

_____

Firma del Autor

Ing. Pedro Manuel Alás Verdecia

_____

Firma del Tutor

Msc. Orlando Arnaldo Valenzuela Aguilera

_____

Firma del Tutor

DATOS DE CONTACTOS

Datos del Tutor:

Nombre y apellidos: Pedro Manuel Alás Verdecia.

Institución: Universidad de las Ciencias Informáticas.

Título: Ingeniero en Ciencias Informáticas.

Correo electrónico: pmalas@uci.cu

Datos del Tutor:

Nombre y apellidos: Orlando Arnaldo Valenzuela Aguilera.

Institución: Universidad de las Ciencias Informáticas.

Título: Máster en Ciencias.

Correo electrónico: ovalenzuela@uci.cu

Datos del Autor:

Nombre y apellidos: Ernesto Cuendias Pérez.

Institución: Universidad de las Ciencias Informáticas.

Correo electrónico: ecuendias@estudiantes.uci.cu

AGRADECIMIENTOS

A mi madre, por ser la persona más importante para mí en este mundo, porque siempre confió en mí y sin su apoyo es muy probable que no estuviera discutiendo mi tesis hoy.

A mi padre, por los consejos que siempre me ha dado, que a pesar de que muchas veces piense que son un poco antiguos, me han sido útiles. Y aunque quizás no te lo demuestre de la mejor forma, eres muy importante para mí.

A la ingeniera más linda que tiene esta universidad, mi Rocío, por darme su amor, su fuerza, subirme los ánimos y pelearme cuando hago mal las cosas, pero principalmente por creer siempre en mí cuando incluso yo dejé de hacerlo.

A mi padrastro y a mi hermana Lupe, porque siempre han estado pendientes de mí y me han apoyado en todo cuanto han podido.

A mis amigos Félix, Manuel, Jorge Ariel, los jimas, Lilian y Araí, por no permitir que me rindiera cuando di todo por perdido en varias ocasiones.

A Yadirka, por arriesgarse y ofrecerme una mano amiga cuando pasaba por momentos difíciles.

A toda la brigada 3503, por recibirme con amor y darme siempre su apoyo y amistad, a pesar de incorporarme en la mitad del segundo año de la carrera.

A mis tutores Pedro y Orlando, por enseñarme a valerme por mí mismo en la parte profesional y a siempre buscar variantes ante los problemas.

A los profesores que he conocido en todo este tiempo y me transmitieron sus conocimientos, convirtiéndose algunos de ellos en buenos amigos.

DEDICATORIA

Dedico este trabajo de diploma a la memoria de mis abuelos Lola y Antonio, por estar siempre a mi lado y darme su apoyo y amor incondicional. Por ser mis segundos padres, compartiendo junto a mí las alegrías y las tristezas. Donde quiera que se encuentren, quiero que sepan que los amo y que siempre vivirán en mi corazón.

RESUMEN

En el Departamento de Desarrollo de Componentes del Centro de Informatización de Entidades fue desarrollado un componente para la gestión de las notificaciones, el cual puede integrarse con varios sistemas para realizar esta función. Los sistemas Xedro-Orbita, Xedro-SIPAC y Sauxe utilizaron este componente, pero de forma personalizada, representando esto una solución a la medida e implicando alta dependencia entre componentes; obligando a que se escriban instrucciones en el código de los componentes que necesiten gestionar las notificaciones. Todo ello incurre en la invariabilidad, poca configuración y la dificultad de selección de un usuario específico para notificarle la ocurrencia de un evento a través de una notificación, lo cual evidencia la necesidad de realizarle mejoras al componente Notificaciones que se utiliza actualmente.

En el presente trabajo de diploma se proponen mejoras funcionales al componente Notificaciones, incorporándole el envío de alertas mediante reglas de negocio a nivel de usuarios, lo que posibilitará incrementar la usabilidad, interoperabilidad e independencia de los componentes que requieran utilizarlo.

Las pruebas de software permiten evaluar la calidad de un producto de software o mejorarlo, mediante la identificación de sus defectos y problemas, para la validación de la solución propuesta se realizaron pruebas de caja blanca, caja negra y aceptación con el cliente.

Palabras claves

Componente, notificación, regla de negocio.

ABSTRACT

In the Component Development Department of the Center of Informatization of Entities it was developed a component for managing notifications, which can be integrated with various systems to perform this function. The systems: Xedro-Orbita, Xedro-SIPAC and Sauxe used this component, but personalized way, this represents a solution to measure and involving high dependence between components; forcing instructions to be written in the code of the components that need to manage notifications. All this incurs on invariance, little configuration and the difficulty of selecting a specific user to notify the occurrence of an event through a notification, which shows the need to make improvements to the Notifications component used today.

In this diploma thesis are proposed functional improvements to the Notifications component, incorporating alerting by business rules at the user level, which will enable to increase the usability, interoperability and independence of the components that require use.

Software testing allow evaluating the quality of a software product or better, by identifying their shortcomings and problems, for validation of the proposed solution white box testing, black box and customer acceptance were made.

Keywords

Component, notification, business rule.

Índice

Índice de figuras

Índice de tablas

INTRODUCCIÓN

La era actual es conocida como la “era de la información”; la misma impone a las personas, como objetivo principal, la necesidad de incrementar los volúmenes de información que se necesitan para subsistir e incluso poder ser competitivo. En este aspecto, la rapidez con la cual se es capaz de obtener la información juega un papel muy importante, por lo que puede llegar a marcar la diferencia en un ámbito en el cual el más informado tiene ventaja sobre los demás.

Hoy en día el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC), dedicados al estudio del tratamiento de la información y la comunicación a través del empleo de sistemas automáticos, ha provocado profundos cambios en la estructura social, laboral e industrial de la sociedad, propiciando con ello el surgimiento de empresas y compañías dedicadas a la producción de software. Esta situación ha sentado base para la aparición de nuevos conceptos, normas, estándares, técnicas y herramientas que han dejado atrás a muchos otros por ser obsoletos cuyo dominio ha tenido un impacto significativo en el desarrollo de las empresas.

Para Cuba ha sido un reto no quedar atrás ante estos avances por lo que aprovechando el potencial profesional con el que cuenta, ha creado organismos, empresas y entidades para solventar la necesidad creciente que existe de desarrollar productos informáticos, tanto para la comercialización como para satisfacer la demanda nacional e internacional. Entre las entidades productoras de software se encuentra la Universidad de Ciencias Informáticas (UCI), creada en el programa de Batallas de Ideas impulsado por nuestro comandante Fidel Castro Ruz en el año 2002, en la que existen varios centros productivos donde se desarrollan un gran número de proyectos de software, entre los cuales se figura el Centro de Informatización de Entidades (CEIGE).

Uno de los productos desarrollados en dicho centro es el marco de trabajo para el desarrollo de aplicaciones web de gestión (Sauxe), el cual contiene componentes y elementos reutilizables que proveen una estructura genérica para lograr mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo (Díaz, 2012). Sauxe posee un componente de gestión de notificaciones que brinda a los usuarios del mismo la posibilidad de conocer una determinada información en el instante en la que se genera o cambia.

Sin embargo, para emitir los avisos haciendo uso del componente Notificaciones de Sauxe, es necesario acceder al código fuente de los otros componentes que necesiten gestionar las

notificaciones y escribir una serie de instrucciones para especificar el evento que las generará. Esto es complejo, pues requiere de conocimientos previos de las soluciones de los componentes, además restringe la configuración a situaciones estáticas y no se ajusta a la dinámica de las notificaciones. Además las notificaciones sólo se llevan hasta el nivel de roles por lo que se persigue extender las mismas hasta el nivel de usuarios. Se desea también seleccionar los elementos comunes en los sistemas SIPAC, Sauxe y Órbita e integrarlos en un solo componente y que este permita configurar gráficamente todo el proceso de notificaciones sin que sea necesario modificar el código fuente de los componentes que necesiten la gestión de notificaciones.

Teniendo en cuenta las dificultades anteriormente planteadas, se plantea como **problema a resolver:** ¿Cómo mejorar la configuración del componente de gestión de notificaciones del marco de trabajo Sauxe para incrementar la usabilidad, interoperabilidad e independencia de las soluciones que lo utilicen?

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:** componente para la gestión de notificaciones del marco de trabajo Sauxe.

Quedando definido como **objetivo general:** incorporar al componente de gestión de notificaciones del marco de trabajo Sauxe el envío de alertas mediante reglas de negocio a nivel de usuarios.

Para darle solución al objetivo general se definen los siguientes **objetivos específicos:**

1. Elaborar el Marco Teórico de la investigación para sustentar los conceptos y la propuesta de desarrollo del componente de gestión de notificaciones del marco de trabajo Sauxe.
2. Realizar el análisis y diseño de las nuevas funcionalidades del componente de gestión de notificaciones del marco de trabajo Sauxe, para el envío de alertas mediante reglas de negocio hasta el nivel de usuarios.
3. Permitir configurar reglas de negocio en el envío de alertas a nivel de usuarios al componente de gestión de notificaciones del marco de trabajo Sauxe.
4. Validar la solución propuesta mediante pruebas de unidad, sistema y aceptación.

Se definen además, las siguientes **tareas de investigación** a realizar para dar cumplimiento a los objetivos específicos:

- Revisión bibliográfica.
- Elaboración de la fundamentación teórica.
- Descripción de la metodología y de las herramientas.
- Realización de los productos de trabajo propuestos por la metodología.
- Implementación de las funciones asociadas a la solución.
- Validación del resultado obtenido.

Teniendo como **idea a defender**: si se incorpora al componente de Notificaciones del marco de trabajo Sauxe el envío de alertas mediante reglas de negocio a nivel de usuarios, será posible incrementar la usabilidad, interoperabilidad e independencia de los componentes que requieran gestionar notificaciones.

Los **métodos teóricos** utilizados en el desarrollo de la investigación son:

- **Análisis Histórico-Lógico:** propició el análisis de la evolución y desarrollo de sistemas de notificaciones para determinar las características y requisitos candidatos en la solución.
- **Analítico-Sintético:** permitió estudiar soluciones existentes a profundidad para seleccionar las características y funcionalidades que se adapten mejor a la solución.

Los **métodos empíricos** utilizados en el desarrollo de la investigación son:

- **Entrevista:** utilizada para obtener información de los especialistas de áreas del CEIGE con el fin de hacer un levantamiento de requisitos y conocer las posibles características y mejoras del componente de gestión de notificaciones.
- **Observación:** empleado durante la realización de análisis de algunas herramientas que se utilizan para el envío de notificaciones en tiempo real.

El documento presenta la siguiente estructura:
Capítulo 1: Fundamentación Teórica: en este capítulo se realiza un estudio del estado del arte sobre las técnicas de notificaciones mediante reglas de negocio empleadas en aplicaciones web, definiéndose algunos conceptos importantes para la investigación. Además se describen tanto la metodología, como las tecnologías y herramientas definidas por la UCI y el Departamento de

Desarrollo de Componentes respectivamente para el desarrollo de sistemas y aplicaciones.

Capítulo 2: Modelado del negocio, análisis y diseño: en este capítulo se describen los requisitos funcionales y no funcionales con los que contará la extensión del componente para la gestión de notificaciones en el marco de trabajo Sauxe. Se describen los patrones de diseño y arquitectónicos definidos para el desarrollo de la solución, elaborándose el diagrama de clases del diseño y el modelo de datos correspondientes a la solución, por último se valida el diseño mediante el empleo de métricas.

Capítulo 3: Implementación, pruebas y validación de la solución propuesta: en este capítulo se especifican aspectos de interés para el proceso de implementación tales como los estándares de codificación del sistema, la descripción de los algoritmos relacionados con las reglas de negocio, los resultados de la validación de la mejora del componente mediante las pruebas de software realizadas y finalmente se valora qué beneficios trae el resultado obtenido en dichas pruebas.

Capítulo I: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En este capítulo se realizará una conceptualización de los términos más importantes referentes a la gestión y envío de notificaciones. Se hará un análisis de sistemas existentes para enviar notificaciones con el fin de identificar elementos que puedan ser reutilizados en el desarrollo de la mejora del componente informático que da solución al problema planteado en la presente investigación. Además serán descritas las herramientas, bibliotecas, lenguajes de programación y marcos de trabajo que se utilizarán durante la implementación.

1.2. Análisis bibliométrico y documental

En esta sección se realiza un análisis de la documentación consultada, con el objetivo de mostrar la novedad y fiabilidad de la revisión bibliográfica basándose en las fechas de las publicaciones. Las bases de datos utilizadas son la de la IEEE y la de Google Scholar. Además, se relacionan los tipos de fuentes bibliográficas más citadas: libros, informes, sitios web y artículos. El análisis realizado se muestra en la siguiente tabla.

Tabla 1: Análisis bibliométrico y documental

Tipo de fuente bibliográfica	**Cantidad consultada**	**Cantidad publicada en los últimos cinco años (2010-2015)**
Libro	12	3
Informe	11	10
Sitio web	19	13
Artículo	4	1
Total:	**46**	**27**

De la tabla anterior se obtiene como resultado que el 58,7% de la literatura consultada pertenece a los últimos cinco años, por lo que se evidencia la actualidad de la bibliografía consultada.

1.3. Conceptos relacionados con las notificaciones en tiempo real

Para ayudar a la comprensión de los términos abordados en la investigación se definen los siguientes conceptos, que tienen una estrecha relación con el problema en cuestión y su solución. Los criterios planteados serán asumidos por el autor del presente trabajo:

Notificación: acción y efecto de comunicar formalmente a su destinatario la ocurrencia de un evento (Díaz, 2014).

Evento: eventualidad, hecho imprevisto, o que puede suceder (Colectivo de autores, 2010).

Regla de Negocio: Establecen los procedimientos que deben ser ejecutados y las condiciones que deben ser evaluadas y controladas en el flujo de Proceso. Son dinámicas, están sujetas a cambios en el tiempo y pueden encontrarse en todo tipo de aplicaciones (Bizagi Limited, 2010).

Tiempo Real (Real-Time Computing o RTC): funcionamiento de un sistema en el que los procesos se realizan simultáneamente a los hechos que representan (Colectivo de autores, 2009).

Sistemas de Tiempo Real (STR): es un sistema en el que el tiempo en que se produce su salida es relevante. Esto es debido a que generalmente la entrada corresponde a algún instante del mundo físico y la salida tiene relación con ese mismo instante (Wellings, 1997).

Interoperabilidad: Esfuerzo requerido para acoplar un sistema a otro (Pressman, 2010).

Usabilidad: El grado en que el software es fácil de usar como lo indican los siguientes subatributos: comprensibilidad, facilidad de aprendizaje, operabilidad (Pressman, 2010).

Independencia: Cualidad o condición de independiente (Real Academia Española, 2000).

Teniendo en cuenta el contenido que aborda el presente trabajo, se dio una definición que estuviera más vinculada al mismo.

Independencia: Propiedad que poseen los componentes de ser autónomos para poder ser modificados sin afectar al resto.

1.4. Sistemas de notificaciones existentes

Actualmente existen soluciones informáticas que abarcan dentro de su dominio la generación de notificaciones a nivel de usuario, siendo esta una característica importante a incorporar en el componente de notificaciones del marco de trabajo de Sauxe. Seguidamente se analizan algunas de ellas teniendo en cuenta las variables de usabilidad, interoperabilidad e independencia. Paralelamente a ello se tendrá en cuenta las políticas por la que se rige la universidad y el país, en cuanto a la independencia tecnológica y la utilización de herramientas libres así como la sustitución de importaciones y ahorro de recursos.

1.4.1 Servidor de correos Zimbra

VMware Zimbra Collaboration Server, es un servidor de correo electrónico y una plataforma de

colaboración de código abierto, desarrollado por Zimbra Inc. (subsidiaria de VMware). Soporta correos electrónicos y calendarios a través de una interfaz web basada en AJAX. Incluye capacidades de búsqueda avanzada, calendario compartido y relaciones de fechas. Puede definir filtros para administrar los mensajes de correo electrónico entrantes y salientes, incluidos los mensajes relacionados con el Calendario y Flujos de actividad. Un filtro consiste en una o más condiciones y una o más acciones. Puede basar una condición en un tema, los mensajes de correo electrónico de una persona específica, o mensajes para un en particular la fecha. Si el mensaje coincide con las condiciones, las acciones especificadas se llevan a cabo.

Se pueden crear filtros haciendo lo siguiente (Zimbra Inc., 2014):

- Ordenar los mensajes entrantes en carpetas. Por ejemplo, todos los mensajes de correo electrónico de su supervisor se mueven automáticamente a la carpeta Gestión Directivas en su recibo.
- Mensajes con etiqueta. Por ejemplo, los mensajes de correo electrónico de las empresas-eventos lista de correo están marcados con la etiqueta de Eventos.
- Aceptar o descartar correo electrónico. Por ejemplo, todos los mensajes de correo electrónico recibidos de una particular dirección de correo electrónico, se mueven a la carpeta Papelera.

Zimbra es una aplicación que utiliza licencia no privativa, posee un gestor reglas de negocio y las notificaciones que genera alcanzan el nivel de los usuarios. Sin embargo es una solución a la medida ya que es un servidor de correos, no un marco de trabajo o una herramienta para desarrollar aplicaciones, por tanto es una solución que no constituye una base tecnológica para desarrollar proyectos que se separen del dominio de correos. Esto implica que al tratar de adaptar a otro sistema la forma en que notifica, se deberá incluir en ese sistema elementos propios del negocio de Zimbra, debido a que los mismos dependen de su arquitectura. Por todo esto, es una solución difícil de extender.

1.4.2 Sistema de notificaciones de RHODA v2.1.

Sistema para el Repositorio de Objetos de Aprendizaje (RHODA) el cual permite enviar notificaciones con la mayor cantidad de información posible a los usuarios. El sistema envía notificaciones vía correo a los usuarios interesados de la información que deseen conocer; además brinda la posibilidad de suscribirse mediante RSS (Really Simple Syndication) a diferentes opciones para mantener a los usuarios actualizados sin necesidad de que visiten el repositorio, el sistema es desarrollado con el framework Symfony, el cual provee una funcionalidad llamada **send ()** desde la instancia Mailer (Pérez, 2011).

El sistema está desarrollado sobre tecnologías libres, permite enviar correos teniendo en cuenta el usuario de las personas suscritas a los diferentes eventos y hace uso de reglas de negocio para generar las notificaciones.

Sin embargo, a pesar de que el sistema RHODA v2.1 emplea reglas de negocio para generar las notificaciones, las mismas no se adaptan a la dinámica de las notificaciones ya que están definidas de forma estática sin posibilidad de ser modificadas. El usuario sólo se suscribe a la regla y cuando ocurre un evento que cumple con las condiciones que esta plantea, la persona es notificada. Lo anterior implica que se vean afectadas la interoperabilidad y la independencia del sistema, por lo que no es factible utilizarlo para desarrollar la solución de Sauxe; además de ser una solución a la medida, lo que conlleva que para migrar dicho sistema a otro, se deban incluir elementos propios de ese negocio.

1.4.3 Mecanismo para alertas y notificaciones del Sistema de Información Hospitalaria ALAS HIS

Sistema que permite el envío de notificaciones y alertas del Sistema de Información Hospitalaria ALAS HIS; para hacer llegar información de interés personal y administrativo a pacientes y trabajadores dentro y fuera de la institución en el menor tiempo posible vía SMS[1], correo y beeper en tiempo real a grupos de usuarios del sistema, teniendo en cuenta el rol que desempeñan. El sistema está implementado con Java y utiliza la librería Javamail para el envío de notificaciones vía correo; a la vez que sus interfaces fueron creadas con JSF[2]. Con el mecanismo de alertas y notificaciones para el Sistema de Información Hospitalaria alas HIS los usuarios podrán suministrar información vía SMS, correo y beeper en tiempo real a grupos de usuarios del sistema, teniendo en cuenta el rol que desempeñan; enviar un mensaje (Beeper) a todo el personal de guardia en caso de urgencia; notificar al paciente cuando estén listos los resultados de los exámenes realizados vía correo y SMS; archivar trazas de usuarios y envíos realizados como constancia y monitoreo de la información enviada; emitir notificaciones automáticamente al responsable de farmacia cuando un lote de medicamento está próximo a vencerse (Estanque Díaz, y otros, 2013).

En resumen Alas HIS es una aplicación desarrollada sobre tecnologías libres, que notifica a los usuarios de la misma teniendo en cuenta el rol que desempeñan y no hace uso de reglas de negocio para generar las notificaciones. Además es una solución a la medida, lo que conlleva que

[1] Short Message System

[2] JavaServer Faces

para incluir este mecanismo en otro sistema, se deban incluir elementos propios de ese negocio.

1.4.4 Solución para alertas y avisos en el Sistema de Planificación de Actividades

El Sistema de Planificación de Actividades (SIPAC) Xedro-SIPAC desarrollado en la UCI por un grupo de especialistas pertenecientes a CEIGE, constituye una herramienta para la gestión de las actividades a todos los niveles organizacionales. Cuenta con varios módulos encargados de generar las configuraciones necesarias para el seguimiento de las tareas principales en función del cumplimiento de los objetivos de cada entidad, así como la gestión de los posibles involucrados, nomencladores y niveles de subordinación basados en reglas de la compartimentación de la información. Permite interrelacionar objetivos de trabajo y actividades en tiempo real, definir el flujo de información hacia los diferentes niveles a partir de la definición de estados y transiciones, así como la recuperación de la información de acuerdo al modelo de planificación actual (Lorente, 2013).

Posee un componente de gestión de notificaciones para facilitar la realización de los procesos de Ejecución y Control de la Planeación Estratégica y Operativa. El componente fue desarrollado en el lenguaje PHP, utilizando herramientas y tecnologías libres (Lorente, 2013). Notifica a los usuarios del mismo teniendo en cuenta el rol que estos desempeñan y no hace uso de reglas de negocio para generar las notificaciones

Al hacerse una revisión de la forma en que SIPAC notifica, esto arrojó que es un sistema a la medida y que el componente envía avisos a un buzón interno a partir de un grupo muy reducido de eventos ligados a su negocio muy particular. Dicha filosofía ya estaba comprendida dentro del componente de gestión de notificaciones de Sauxe, siendo mucho más genérica y configurable.

1.4.5 Componente de gestión de notificaciones del Sistema de Control de Flota y Mantenimiento Vehicular

El Sistema de Control de Flota y Mantenimiento Vehicular Xedro-Orbita es un sistema de gestión web desarrollado en la UCI por un grupo de especialistas pertenecientes a CEIGE, con el objetivo de informatizar el área de transporte de la universidad. Lleva el control de los vehículos para el mantenimiento preventivo planificado y correctivo. Cuenta con una serie de componentes que permiten realizar de manera informatizada todos los procesos de esa área (Díaz, 2014).

Posee un componente de gestión de notificaciones el cual funciona mediante tareas programadas y permite suscribir los roles del sistema a una o varias notificaciones generadas en los diferentes componentes del mismo (Díaz, 2014). Está desarrollado sobre tecnologías libres, y no gestiona

reglas de negocio para generar las notificaciones.
Además, la generación de las alertas está enmarcada en el funcionamiento específico de Órbita, dependiendo de los módulos y subsistemas que lo conforman, por lo que para incluir este componente en otro sistema, se deban incluir elementos propios de ese sistema.

1.4.6 Marco de trabajo para el desarrollo de aplicaciones web de gestión (Sauxe)

Es un marco de trabajo desarrollado en el Departamento de Desarrollo de Componentes de CEIGE en conjunto con la Unidad de Compatibilización y Desarrollo para la Defensa (UCIFAR), actualmente conocida en el mercado empresarial como la Empresa de Tecnologías para la Defensa (XETID). El mismo contiene componentes y elementos reutilizables que proveen una estructura genérica para lograr mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. Posee las características tecnológicas para la implementación de aplicaciones web de gestión. Cuenta con una arquitectura que permite reutilizar código y promover buenas prácticas como el uso de patrones. Tiene un componente para la gestión de notificaciones pero no posee un gestor de reglas de negocio, afectándose de esta forma la interoperabilidad, la usabilidad y la independencia del mismo. Sauxe está concebido para el desarrollo de aplicaciones web de gestión, por lo que sus componentes y funcionalidades poseen la característica de ser genéricos.

1.5. Análisis valorativo

Con el fin de conocer si alguno de los sistemas estudiados posee elementos que pudieran ser reutilizados en el diseño o la implementación de la extensión del componente de gestión de notificaciones de Sauxe, se establece una comparación entre los mismos teniendo en cuenta los indicadores: tipo de licencias, el nivel que alcanzan las notificaciones, si se gestionan o no reglas de negocio para notificar, y la facilidad de extensión.

Tabla 2: Tabla comparativa entre los sistemas

	Indicadores			
Sistema	**¿Privada o libre?**	**Nivel que alcanzan las notificaciones**	**Gestiona reglas de negocio para notificar**	**Fácil de extender**
Servidor de correos Zimbra	Libre	Usuario	Sí	No
RHODA v2.1	Libre	Usuario	Sí	No

ALAS HIS	Libre	Rol	No	No
Xedro-SIPAC	Libre	Rol	No	No
Xedro-Orbita	Libre	Rol	No	No
Sauxe	Libre	Rol	No	Sí

A partir de la información mostrada en la tabla comparativa, el análisis arrojó como resultado que:

- Todas las aplicaciones son libres de licencia, lo cual es muy positivo ya que la UCI tiene como política el empleo de tecnologías libres.
- Tanto el servidor de correos Zimbra como el sistema RHODA v2.1 emiten las notificaciones teniendo en cuenta los usuarios; mientras que los restantes sistemas lo hacen de acuerdo al rol.
- A excepción del servidor de correos Zimbra y del sistema RHODA v2.1, los restantes sistemas no emplean un gestor de reglas para generar las notificaciones.
- A excepción de Sauxe, los sistemas restantes son sistemas a la medida, por lo que las soluciones que los mismos proponen están estrechamente ligadas al negocio donde se encuentran, complejizando incorporar las mismas a Sauxe.
- Teniendo en cuenta que el Sistema de Control de Flota y Mantenimiento Vehicular fue desarrollado sobre el marco de trabajo Sauxe, serán reutilizados algunos servicios y algunos elementos a nivel de interfaz. De igual forma, se reutilizará la lógica que emplea el servidor de correos Zimbra para la creación de filtros.

1.6. Metodología de desarrollo de software

Se conoce como metodología al conjunto de procedimientos, técnicas, herramientas y al soporte documental que ayuda a los desarrolladores a realizar un software. Una metodología define quién debe hacer qué, cuándo y cómo debe hacerlo (Barreto, 2012).

Se utiliza la definida por la UCI, una variante de la AUP (Proceso Unificado Ágil o Agile Unified Process) ya que es aplicable a todos los proyectos existentes en la universidad y es creada con el objetivo de que exista una estandarización entre todos ellos.

Cuenta con tres fases: Inicio, Ejecución y Cierre; y siete disciplinas: Modelado del negocio (opcional), Requisitos, Análisis y diseño, Implementación, Pruebas internas, Pruebas de liberación, Pruebas de aceptación y Despliegue (opcional) (Universidad de las Ciencias Informáticas, 2014).

Está basada en principios y buenas prácticas propuestos en el Modelo CMMI-DEV v1.3, permitirá elaborar la arquitectura y el diseño del producto, así como generar los productos de trabajo:

modelo conceptual, descripción de requisitos mediante historias de usuarios, diagramas de clases con estereotipos web, diseños de casos de prueba y el manual de usuario (Universidad de las Ciencias Informáticas, 2014). Se generarán sólo estos productos de trabajo ya que la Analista Principal del Departamento de Gestión de Componentes decidió que eran los mínimos necesarios para concluir la documentación del desarrollo de software, decisión aprobada por el Jefe del Departamento.

Para el desarrollo de software se hace necesario utilizar un conjunto de herramientas de apoyo a los procesos definidos en las disciplinas y las fases de la metodología.

1.7. Tecnologías y herramientas

Se describen las tecnologías y herramientas a utilizar durante todo el proceso de implementar la extensión del componente. Las mismas son definidas por el Departamento de Desarrollo de Componentes y se clasifican: lenguajes de programación; bibliotecas; marcos de trabajo; lenguaje, herramienta y notación de modelado; servidores; y protocolos.

1.7.1 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 este pueda comunicarse con el hardware y software existentes (Escribano, 2002). A continuación se describen los lenguajes de programación a utilizados en el desarrollo de la extensión del componente.

PHP v5.3.10

PHP (acrónimo de "PHP: Hypertext Preprocessor") es un lenguaje de código abierto interpretado, de alto nivel, embebido en páginas HTML y el lenguaje está orientado al desarrollo de aplicaciones web, que son interpretadas del lado del servidor. Sus sintaxis son muy similares a lenguajes como C y PERL. Puede ser utilizado en casi todos los sistemas operativos existentes, permitiendo migrar las aplicaciones de un sistema a otro sin necesidad de realizar cambios en el código. Su rapidez en la ejecución y los bajos consumos de prestaciones en los sistemas donde es desplegado, lo hacen uno de los preferidos por los desarrolladores.

Se integra perfectamente a la mayoría de los Sistemas Gestores de Bases de Datos. Su mayor ventaja radica en ser un lenguaje libre, por lo que se convierte en una alternativa de muy fácil acceso, además de poseer una comunidad de desarrolladores que intercambian experiencias, de

esta forma cuando se presenta un problema, es muy fácil obtener documentación para darle solución de forma rápida y sin costo alguno (Colectivo de autores, 2011).

JavaScript v1.10.3

JavaScript es un lenguaje orientado a objetos, que se utiliza principalmente para crear páginas web dinámicas y permite el desarrollo de interfaces de usuario mejoradas. Una página web dinámica es aquella que permite la interacción entre el contenido de la misma y el usuario. JavaScript permite incorporar a dichas páginas efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario. Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios (Flanagan, 2006).

1.7.2 Bibliotecas y Marcos de Trabajo

El proceso de implementación de la solución se efectuará utilizando el marco de trabajo Sauxe, el cual contiene un conjunto de componentes reutilizables que provee la estructura genérica para lograr una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo.

Marco de trabajo para el desarrollo de aplicaciones web de gestión Sauxe v2.2

En el desarrollo de la mejora del componente de gestión de notificaciones utilizará Sauxe como marco de trabajo, el cual posee las características tecnológicas para la implementación de aplicaciones web de gestión. Además cuenta con una arquitectura que permite reutilizar código y promover buenas prácticas tales como el uso de patrones. Por otra parte, Sauxe está compuesto por los marcos de trabajo Extjs para la capa de presentación, Zend para la lógica de negocio y Doctrine para la capa de acceso a datos.

EXTJS v4.2.0

Marco de trabajo que soporta JavaScript para el desarrollo de aplicaciones Web. Actúa del lado del cliente y se utiliza para la creación de las interfaces de usuario y el manejo de sus eventos. Crea todos los objetos HTML (Lenguaje de marcas hipertextuales) a través de la manipulación del Modelo de Objetos del Documento (DOM). Provee un balance entre Cliente-Servidor, ya que distribuye la carga, permitiendo que el servidor gestione más clientes al mismo tiempo. Permite

obtener información del servidor sin estar sujeto a la acción de un usuario (Baryolo Gómez, y otros, 2010).
En el proceso de implementación de la solución se empleará para crear las interfaces que verá el usuario final y con las que interactuará.

Zend Framework v1.12

Brinda soluciones para construir aplicaciones web y servicios modernos, robustos y seguros (Zend Technologies, 2006). Será utilizado dado que se encuentra bajo la licencia Distribución de Software Berkeley (BSD), lo cual permite su distribución, así como las aplicaciones que se desarrollen con él. Proporciona los componentes que forman la estructura del patrón Modelo-Vista-Controlador (MVC).Permite convertir estructuras de datos PHP a la Notación de Objetos de JavaScript (JSON) y viceversa, para su utilización en las aplicaciones.

Doctrine v1.2.2

Doctrine es un marco de trabajo para el mapeo objeto relacional que está dividido en dos capas principales, la capa de abstracción de base de datos (DBAL) y el mapeador de objetos relacionales (ORM) para PHP 5.2 (doctrine.org, 2011).
Se empleará debido a que Sauxe utiliza en la capa de acceso a datos el Lenguaje de Consulta de Datos (DQL) que implementa Doctrine. Una de sus principales características es la opción de escribir las consultas de base de datos en un objeto con una propiedad orientada al lenguaje SQL, llamada DQL. Tiene la ventaja de transformar clases a tablas de una base de datos y viceversa por lo que es empleado para efectuar las consultas a la base de datos (doctrine.org, 2011).

XMPPHP

XMPPHP es una biblioteca que proporciona una solución elegante con un enfoque directo. Permite conectarse a cualquier servidor XMPP 1.0 (Google Talk, Talk LJ, jabber.org). Posee soporte para la Seguridad en la Capa de Transporte (TLS) (Werdmuller, 2010).
Algunas de las características que incluye son:

- Permite conectarse a cualquier servidor XMPP 1.0 (Google Talk, Talk LJ, jabber.org, entre otros).
- Varios enfoques de procesamiento XML[3] y soporte de estilos.
- Se utilizará para la conexión con el servidor Openfire (xmpphp, 2010).

[3] XML(eXtensible Markup Language):Lenguaje de Etiquetado Extensible

Strophejs

Es una colección de bibliotecas JavaScript que se empleará para comunicarse con el protocolo XMPP. Se basa en flujos bidireccionales sobre HTTP[4] sincrónico para emular la persistencia con estado de conexión en ambos sentidos a un servidor XMPP (Moffitt, 2010).

1.7.3 Lenguaje, herramienta y notación de modelado

Notación de Modelado de Procesos de Negocio: BPMN v2.0

BPMN (del inglés Business Process Modeling Notation) es una notación gráfica que describe la lógica de los pasos de un proceso de negocio. Esta notación ha sido especialmente diseñada para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes de las diferentes actividades. Proporciona un lenguaje común para que las partes involucradas puedan comunicar los procesos de forma clara, completa y eficiente. De esta forma define la notación y semántica de un Diagrama de Procesos de Negocio (Business Process Diagram, BPD) (Bizagi Limited, 2010).

Lenguaje Unificado de Modelado: UML v2.0

Es una tecnología orientada a objetos que utiliza un conjunto de símbolos para modelar, construir y documentar gráficamente los diversos componentes del sistema y soporta extensiones personalizadas a través de elementos estereotipados (Object Management Group, 2001)

Visual Paradigm v8.0

Herramienta de Ingeniería de Software Asistida por Computadora (CASE) que: “propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación” (Pressman, 2005).

Soporta los principales estándares como UML, SysML, BPMN 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. Se emplea para la generación de los diagramas del modelado del negocio, la administración de requisitos y el diseño (Visual Paradigm, 2010).

[4] HTTP (Hypertext Transfer Protocol): Protocolo de Transferencia de Hipertextos

1.7.4 Herramientas de desarrollo

Mozilla Firefox v35.0.1

Mozilla Firefox es un navegador web, con interfaz gráfica de usuario desarrollado por la Corporación Mozilla y un gran número de voluntarios externos. Es multiplataforma y está disponible en versiones para Microsoft Windows y Linux. Entre sus principales características se encuentran: barra de direcciones inteligente, Identificación instantánea del sitio web, Anti-malware, Anti-phishing, Control de contenido, Programas antivirus, Gestor de contraseñas, Actualización automática, Bloqueador de ventanas emergentes, Gestor de descargas, Corrector ortográfico, Restauración de sesiones, Sugerencias de búsqueda y búsqueda en la web integrada (Mozilla, 2010).

IDE NetBeans v7.4

El IDE NetBeans es un entorno de desarrollo integrado disponible para Windows, Mac, Linux y Solaris. Consiste en un IDE de código abierto y una plataforma de aplicaciones que permiten a los desarrolladores crear rápidamente aplicaciones web, empresariales, de escritorio y aplicaciones móviles utilizando la plataforma Java, así como PHP, JavaScript y Ajax, entre otros. Se integra con el control de versiones Subversion (NetBeans.org, 2013).

Herramienta para el control de versiones Subversion v1.6.6

Es un sistema de control de revisiones de código abierto, multiplataforma y que está distribuido bajo licencia Apache. Se basa en un repositorio central: es como un servidor de archivos pero con la peculiaridad de que recuerda todos los cambios que se hayan hecho a los archivos y directorios. De este modo se puede volver a versiones antiguas de los mismos y ver cuándo se hicieron los cambios y quién los realizó (CollabNet Inc., 2010).

1.7.5 Servidores

Servidor de mensajería instantánea Openfire v3.7.1

Es un servidor de mensajería para establecer conexiones servidor a servidor que permitan compartir sus usuarios. Utiliza el protocolo Jabber, permite añadir complementos y cuenta con una interfaz web de administración (Ignite REALTIME, 2008).

Servidor de aplicaciones Apache v2.2.22

Es un servidor HTTP libre, open source y su primera versión fue en el año 1995 (como sustituto del httpd de NCSA[5]). Es ejecutable en varios sistemas operativos: Unix, FreeBSD, Linux, Solaris, Novell NetWare, OS X, Microsoft Windows. Permite múltiples lenguajes de script: PHP, Perl, Tcl, Python. Brinda la posibilidad de crear Virtual Host, es decir un mismo servidor físico para varias IPs y nombres de dominio (Mestras, 2012-2013).

Servidor de base de datos PostgreSQL v9.1

PostgreSQL es un potente sistema de gestión de bases de datos objeto-relacional (O-RDBMS), multiusuario, centralizado y de propósito general. Es compatible con los principales Sistemas Operativos: Linux, Unix, Mac OS y Windows (Lopez, 2013).

1.7.6 Protocolos

Protocolo XMPP/Jabber

Es un protocolo abierto y extensible basado en XML, originalmente ideado para mensajería instantánea. Es usado además en una amplia gama de aplicaciones de mensajería de voz y video (xmpp.org, 2010).

Entre sus características aparecen las siguientes (xmpp.org, 2010):

- Abierto: el protocolo es gratuito y de fácil comprensión. Es por ello que cuenta con múltiples implementaciones entre clientes, servidores, componentes de servidores y librerías de código.
- Estándar: el Internet Engineering Task Force (IETF) ha formalizado el núcleo del protocolo como una tecnología de mensajería instantánea e información de presencia.
- Probado: las primeras tecnologías fueron desarrolladas en 1998 y ahora son muy estables. Existen miles de servidores utilizando este protocolo en Internet, y millones de personas utilizándolo para mensajería instantánea para servicios públicos como Google Talk e implementaciones en organizaciones.
- Flexible: las aplicaciones originales de XMPP (de mensajería y presencia) se han extendido y ahora pueden encontrarse en administración de redes, sindicalización de contenidos, herramientas de colaboración, compartimiento de archivos, juegos, monitoreo

[5] NCSA (National Center for Supercomputing Applications): Centro Nacional para Aplicaciones de Supercomputación

de sistemas remotos, servicios web, computación en la nube, etc.

1.8. Conclusiones parciales

En el desarrollo de este capítulo se hizo un estudio de los sistemas que gestionan notificaciones, lo que permitió obtener las funcionalidades deseables teniendo presente las variables: usabilidad, interoperabilidad e independencia. Además se definieron las herramientas, para los procesos de modelado y desarrollo de la aplicación.

Capítulo II: MODELADO DEL NEGOCIO, ANÁLISIS Y DISEÑO

2.1 Introducción

En este capítulo se presenta la propuesta de solución para la extensión del componente de gestión de notificaciones del marco de trabajo Sauxe. Se confecciona el modelo conceptual, además se identifican y describen tanto los requisitos funcionales como los no funcionales. Como productos de trabajo se generan los diagramas de secuencia y de clases del diseño, que incluye los mecanismos y patrones utilizados. Finalmente se aplican las métricas seleccionadas para evaluar el diseño.

2.2 Modelo conceptual

Un modelo conceptual o modelo de dominio, constituye una representación visual para el usuario de los conceptos u objetos significativos del mundo real para un problema o área de interés. Representa conceptos del mundo real, no de los componentes de software, mediante clases conceptuales del dominio del problema, encargándose de capturar los tipos más importantes de objetos y eventos que suceden en el entorno (Pressman, 2005).

Se representa como un diagrama de clases en el que se muestran conceptos u objetos del dominio del problema, así como las relaciones entre las clases conceptuales. Se realiza para facilitar la comunicación entre analistas y clientes, permitiendo comprender las necesidades del usuario y los requisitos del software.

En la Figura 1 se muestra el modelo conceptual de la propuesta de solución, en el mismo se define que: los sistemas tienen componentes, usuarios y roles. Ellos pueden ser suscriptores por lo que pueden recibir notificaciones vía jabber y correo electrónico. Dichas notificaciones llegarán a los suscriptores cuyos usuarios están contenidos en el servidor de mensajería Openfire. El componente gestiona un catálogo el cual tiene todos los eventos o acciones que ocurren en el sistema y los suscriptores. Dichos eventos conforman las reglas de negocio que generan las notificaciones.

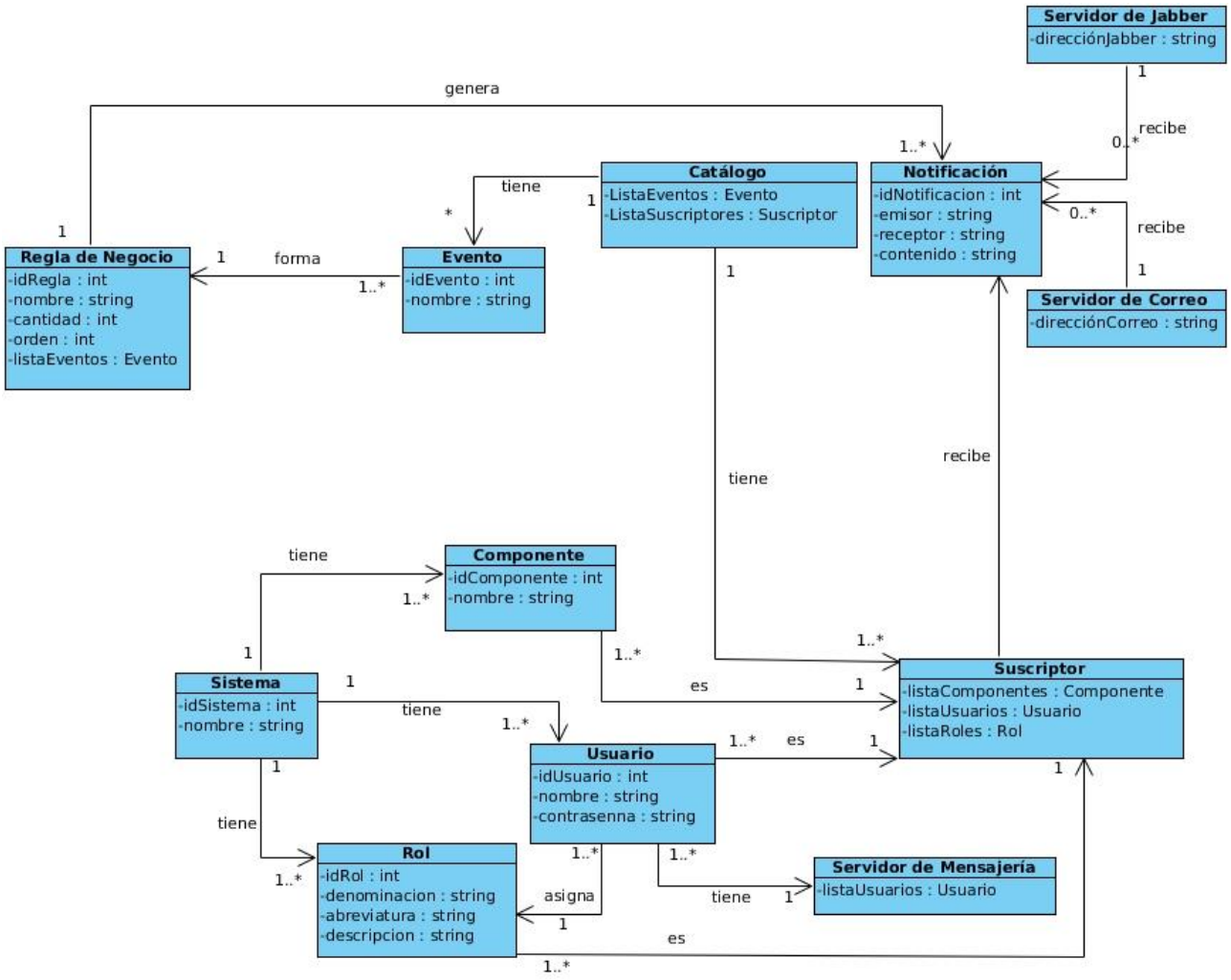


Figura 1: Modelo conceptual

Cada concepto definido contiene un diccionario de datos que expone una breve descripción de los mismos, además de mostrar para cada atributo información relevante como el tipo de datos y las restricciones. Para obtener mayor información sobre el modelo conceptual consultar el documento entregable: **Modelo Conceptual_Notificación Sauxe.odt** que se encuentra en el expediente de proyecto.

2.3 Requisitos

Un requisito del software es una característica que se debe exhibir por el software desarrollado o adaptado para solucionar un problema particular (Pressman, 2005). Estos se dividen en dos grupos: funcionales y no funcionales.

Los **requisitos funcionales** son las declaraciones de los servicios que el sistema debe

proporcionar, cómo el sistema debe reaccionar a entradas específicas y cómo el sistema debe comportarse en situaciones particulares. En algunos casos, los requisitos funcionales también pueden declarar explícitamente lo que el sistema no debe hacer (Sommerville, 2005).

Con el fin de comprender lo que desea el cliente y obtener una solución que lo satisfaga, se realiza la captura, especificación, administración, priorización y validación de los requisitos del software. Estas actividades se describen a continuación.

2.3.1 Técnica de captura de requisitos

A continuación se explican brevemente en qué consisten las técnicas de captura de requisitos que fueron empleadas.

- **Tormenta de ideas:** consiste en la acumulación de ideas sin prejuicios y valoraciones que puedan descartarlas y aunque no evidencia los detalles concretos que se pueden necesitar del sistema, es muy común en los comienzos del proceso de ingeniería de requisitos (Escalona , y otros, 2002).
- **Uso de prototipos:** los prototipos son sistemas teóricos o desarrollados que nos permiten presentar las ideas de lo que queremos realizar al cliente para su validación. De la observación del prototipo se obtendrán nuevos requisitos para completar la lista (Colectivo de autores, 2010).

Como resultado de aplicar las técnicas estudiadas se identificaron los requisitos funcionales que se agruparon de la siguiente forma:

R.F-1.0: Gestionar notificaciones.

R.F-1.1: Asociar usuario a notificaciones.

R.F-1.2: Desasociar usuario de notificaciones.

R.F-2.0: Gestionar reglas de negocio.

R.F-2.1: Adicionar regla de negocio.

R.F-2.2: Modificar regla de negocio.

R.F-2.3: Eliminar regla de negocio.

R.F-2.4: Asociar regla de negocio a una notificación.

R.F-2.5: Listar reglas de negocio.

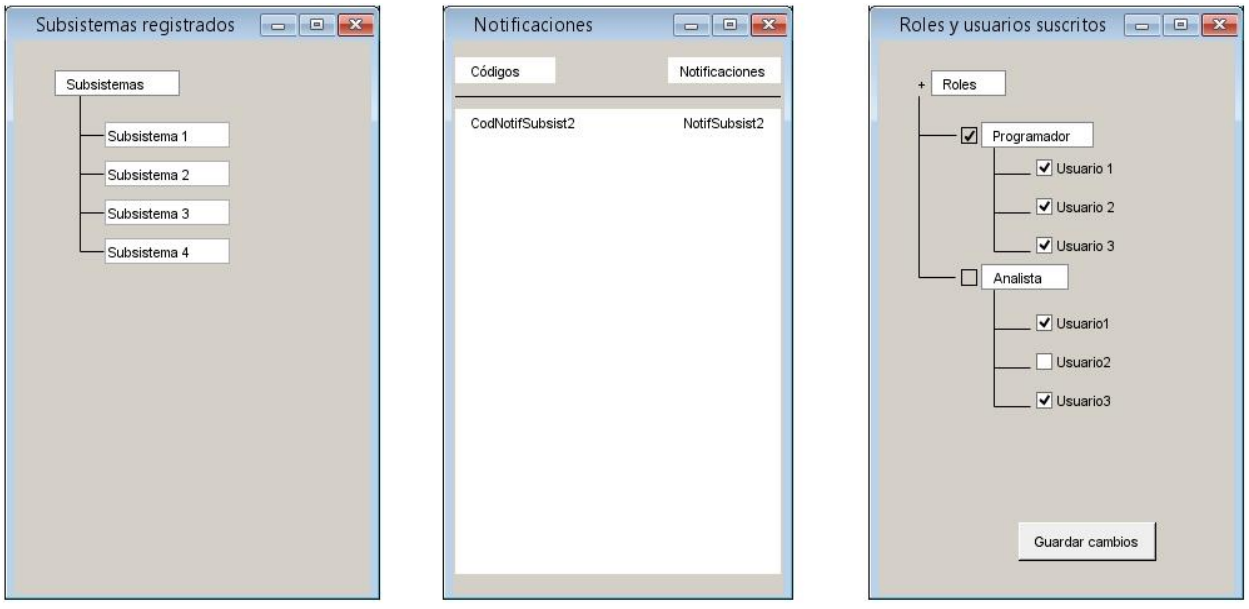
R.F-3.0: Chequear cumplimiento de regla.

2.3.2 Descripción de los requisitos funcionales

Una vez identificados los requisitos se realizó su descripción siguiendo la plantilla de Historias de Usuarios. La tabla 3 muestra la descripción del requisito Asociar usuario a notificaciones debido a

que en la versión actual del componente las mismas llegan sólo hasta el nivel de roles, por lo que no es posible determinar a qué personas específicas llegará o dejará de llegar una alerta.

Tabla 3: Descripción del requisito Asociar usuario a notificaciones.

Historia de usuario	
Número: 1.1	**Nombre del requisito:** Asociar usuario a notificaciones
Programador: Ernesto Cuendias Pérez	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 10 días
Riesgo en Desarrollo:	**Tiempo Real:** 2 horas
Descripción: El sistema debe permitir asociar uno o varios usuarios a alguna notificación. Para ello se selecciona alguno de los componentes registrados en el sistema. Se selecciona alguna de las notificaciones asociadas a ese componente. Se selecciona el árbol **Roles** y se selecciona el rol o los roles que se deseen notificar. Se da clic en el botón **Guardar cambios**. Se debe mostrar un mensaje: “Se han guardado los cambios”. Si la asociación se desea hacer a nivel de usuarios, se despliega el rol deseado y se selecciona el usuario o los usuarios que se deseen notificar. Se da clic en el botón **Guardar cambios**. Se debe mostrar un mensaje: “Se han guardado los cambios”.	
Observaciones: se asoció al menos un nuevo usuario a una notificación.	
Prototipo de interfaz: 	

Las descripciones de los requisitos restantes se encuentran en los documentos: **Historias de usuario_Gestionar usuario.odt** e **Historias de usuario_ Gestionar regla de negocio.odt**, en el expediente de proyecto.

2.3.3 Validación de los requisitos

Esta actividad se realizó con el objetivo de garantizar que los requisitos fueran correctos y cumplieran con las necesidades del cliente. Se desarrolló mediante las siguientes técnicas:

- **Validación de requisitos mediante Prototipos.**

Se presentaron los prototipos de interfaz de usuario elaborados durante la especificación a especialistas funcionales, para corroborar su correspondencia con las necesidades y aspiraciones del cliente. Para ello, se desarrollaron varios escenarios posibles con el auxilio de juegos de datos, de forma tal que se visualizaran las diferentes funcionalidades que tendría el sistema. En un total de tres iteraciones fueron encontradas 4 No Conformidades (NC) que fueron corregidas, siendo su distribución de la siguiente forma: Iteración1: 3 NC, Iteración 2: 1NC, e Iteración 3: 0 NC.

- **Generación de Diseños de Casos de Prueba.**

La generación de casos de prueba tiene como objetivo comprobar la veracidad de los requisitos funcionales (Sommerville, 2005) y para cada uno de ellos se definieron los datos de entrada, las tareas a realizar y los resultados esperados.

2.3.4 Requisitos no funcionales

Los **requisitos no funcionales** son restricciones sobre los servicios o funciones ofrecidas por el sistema. Incluyen restricciones de tiempo, las limitaciones en el proceso de desarrollo y estándares. Los requisitos no funcionales a menudo se aplican al sistema como un todo. Por lo general no sólo se aplican a funciones o servicios del sistema individuales (Sommerville, 2005). Para la aplicación fueron capturados los siguientes requisitos no funcionales:

Usabilidad:

- Eliminar el acceso al código para configurar la notificación, definiendo reglas de negocio para facilitar el uso del componente.
- Señalizar errores cometidos por el usuario.
- Desactivar opciones que requieran de condiciones previas para su activación.
- Extender las notificaciones hasta el nivel de usuario, gestionando la asociación de roles y usuario a una notificación mediante un árbol.

Seguridad:

- Garantizar la protección e integridad de los datos, utilizando el sistema Acaxia.
- Garantizar que la información sólo llegue a la persona indicada, definiendo el acceso a la misma hasta el nivel de usuario.

Interoperabilidad:

- Mejorará entre los componentes ya que anteriormente existía dependencia del código directamente pues cada componente tenía que hacer uso de un servicio para lanzar la notificación. En el presente todo el proceso se configura desde el componente de gestión de notificaciones y no hay que modificar el código dentro del componente; sólo se asocia una notificación a una regla, la cual está formada por los eventos que ocurren en el sistema.
- Eliminar la dependencia entre componentes que requieran la gestión de notificaciones, configurando todo el proceso de manera gráfica a través de reglas de negocio.

Mantenibilidad:

- Eliminar el efecto de los cambios que se produzcan en las notificaciones, sobre el resto de los componentes.

Hardware:

Los requerimientos recomendados para el correcto funcionamiento de la aplicación son los siguientes:

- Memoria RAM de 2GB o superior.
- Disco duro de 80GB o superior.
- Microprocesador 2.2 GHz o superior.

Software:

- Sistema operativo Linux en cualquiera de sus distribuciones.
- Navegador web Mozilla Firefox 35.0.1 o superior.
- Servidor Apache 2.2.
- Servidor de base de datos PostgreSQL 8.3.
- Biblioteca PHP5.
- Openfire 3.7.1

2.4 Diseño

Los contenidos que a continuación se abordan constituyen los productos de trabajo que conforman el modelo de diseño de la solución. El modelo de diseño es el punto de partida para una correcta implementación y constituye un plano de la misma. Además toma los requisitos como

entrada para establecer los mecanismos de diseño, los diagramas de clases del diseño, los patrones de diseño, diagramas de secuencia y modelo de datos.

2.4.1 Patrón arquitectónico Modelo-Vista-Controlador (MVC)

Para el desarrollo de la extensión del componente de gestión de notificaciones se hace uso del patrón arquitectónico MVC, ya que está implementado dentro de Sauxe, sobre el cual será desarrollada la solución. En el cual se permite separar la interfaz de usuario, el control y los datos en agrupaciones denominadas modelos, vistas y controladoras. **La Vista** visualiza las interfaces de usuario y la respuesta a las peticiones hacia el controlador. **El Controlador** está constituido por las clases controladoras que permiten la interacción entre la vista y el modelo, administrando lo eventos que se generan en el sistema. **El Modelo** contiene las clases encargadas de la lógica del negocio y las que se dedican al acceso a datos de la base de datos.
Una vez definido el patrón arquitectónico, es necesario determinar los patrones de diseño a emplearse, los cuales se describen a continuación.

2.4.2 Patrones de diseño

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software (Larman, 2003). Brindan una solución ya probada y documentada ante dificultades en el desarrollo de software que estén sujetas a contextos similares. Su uso pretende establecer un lenguaje común entre los programadores, contribuir a la reutilización, ahorrar tiempo en la implementación y obtener un producto con calidad.
En la solución fueron empleados dos tipos de patrones de diseño, los Patrones generales de software para asignar responsabilidades (**GRASP**) y los Patrones del Grupo de los Cuatro (**GoF**), descritos a continuación:
GRASP: es un acrónimo que significa General Responsibility Asignment Software Patterns. Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos (Visconti , y otros, 2004). Estos patrones son: Experto, Bajo Acoplamiento, Alta Cohesión, Controlador y Creador.

- El patrón **Experto** asigna una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad (Visconti , y otros, 2004). Se evidencia en la clase DatEventoNotif pues la misma cuenta con la información necesaria para cumplir con las responsabilidades asignadas.

```php
class DatEventoNotif extends BaseDatEventoNotif
{

    public function obtenerCantidadDadoIdAccion($idaccion, $idregla)
    {
        $q = Doctrine_Query::create();
        $result = $q->select('*')
            ->from('DatEventoNotif')
            ->where('idaccion =? and idregla = ?', array($idaccion,$idregla))
            ->setHydrationMode(Doctrine::HYDRATE_ARRAY)
            ->execute();
        return $result;
    }
```

Figura 2: Fragmento del código donde se evidencia el patrón Experto

- El empleo del **Bajo Acoplamiento** garantiza que las clases se encuentren lo menos relacionadas entre sí, para que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión en las demás, potenciando la reutilización y disminuyendo la dependencia entre las clases (Lorente, 2013). Existen el número mínimo de relaciones para satisfacer las necesidades del componente.
- El patrón **Alta Cohesión** garantiza que cada clase realice sólo las funciones que estén en correspondencia con la responsabilidad que posea (Lorente, 2013). Existen las funciones suficientes en las clases para garantizar el cumplimiento de las necesidades del componente.
- El patrón **Controlador** fue empleado para la creación de las clases controladoras que intervienen en los diferentes procesos, ejemplo de ello es AMELReglaDeNegocioController.

```
class AMELReglasDeNegocioController extends ZendExt_Controller_Secure
{

    public function init()
    {
        parent::init();
    }

    public function AMELReglasDeNegocioAction()
    {
        $this->render();
    }

    public function cargarReglasAction()
    {
        $limit = $this->_request->getPost('limit');
        $start = $this->_request->getPost('start');

        $model = new GestionarReglaNegocioModel();
        //////**********************
        $model->ejecutarReglaDeNegocioParaNotificar('18');
        //$model->ejecutarReglaDeNegocioParaNotificar('8');
        //$model->ejecutarReglaDeNegocioParaNotificar('10');

        return $model->listarReglas($limit, $start);

    }

    //carga las acciones que forman una regla especifica
    public function cargarAccionesQueTieneReglaAction()
    {
        $limit = $this->_request->getPost('limit');
```

Figura 3: Fragmento del código donde se evidencia el patrón Controlador

- El patrón **Creador** permite la creación de instancias, donde la nueva clase deberá ser creada por la que tiene toda la información necesaria para realizar esa acción (Visconti , y otros, 2004). Este patrón se puede observar en la clase AMELReglaNegocioModel, quien es la responsable de crear objetos del modelo para acceder a las clases correspondientes.

```
class AMELReglaNegocioModel extends ZendExt_Model
{
    public function AMELReglaNegocioModel()
    {
        parent::ZendExt_Model();
    }

    public function listarReglas($limit, $start)
    {
        $datRegla = new DatReglaNotif();
        $accionesReglas = $datRegla->listarRegla($limit, $start);
```

Figura: Fragmento del código donde se evidencia el patrón Creador

GoF: los patrones Gang Of Four (GoF) son patrones de diseño publicados en el libro Design Patterns: 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/u objetos.
- **De Comportamiento:** caracterizan las formas en las que interactúan y reparten responsabilidades las distintas clases u objetos (Unidad docente de Ingeniería del software).

De los patrones GoF existentes a continuación se describen los que fueron utilizados para la extensión del componente:

- El patrón **Fachada** tiene como propósito proporcionar una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. La "fachada" satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas (Unidad docente de Ingeniería del software). Para acceder a las funciones que están en otro componente se utilizan las funciones de inversión de control (IOC). Sauxe tiene definida una clase para acceder como fachada a las funciones de menor nivel de los componentes que les interese a un tercero.
- El patrón **Singleton** tiene como propósito garantizar que una clase sólo tenga una única instancia, proporcionando un punto de acceso global a la misma (Unidad docente de Ingeniería del software). Este patrón se puede observar en la función insertarNotifBus de la clase DatMessagebusModel.php, donde se crea una única instancia de Doctrine_Manager::getInstance() para establecer la conexión con la base de datos.

```
public function insertarNotifBus(DatMessagebus $datMessageBus)
{
    try {
        $mb = new DatMessagebus();
        $iddat_messagebus = $mb->existe($datMessageBus);
        if ($iddat_messagebus == null) {
            $dm = Doctrine_Manager::getInstance();
            $conn = $dm->getCurrentConnection();
            $conn->beginTransaction();
            $datMessageBus->save($conn);
            $conn->commit();

        }
    } catch (Exception $exc) {
        echo $exc->getTraceAsString();
    }
}
```

Figura 4: Fragmento del código donde se evidencia el patrón Singleton

2.4.3 Mecanismos de diseño

En el diseño de la solución se hizo uso de varios mecanismos de diseño con el objetivo de simplificar los diagramas de clases permitiendo mejorar la comunicación entre los miembros del equipo de desarrollo. Permiten simplificar el tiempo de asimilación del diseño por parte de los desarrolladores.

Mecanismo de diseño para los nombres de las clases.

Los diagramas de clases del diseño de la extensión del componente de gestión de notificaciones se realizaron por agrupaciones de requisitos funcionales. En el caso de las agrupaciones de requisitos conformadas por adicionar, modificar y eliminar se agrupan en una clase a la cual se le denomina AMEL_Nombre de la clase.

Figura 5: Mecanismo de diseño para los nombres de las clases

2.4.4 Diagrama de clases del diseño

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de

software y de las interfaces en una aplicación, contiene información como asociaciones, atributos, métodos, dependencias (Visconti, y otros, 2004).

En la Figura 6 se muestra el diagrama de clases del diseño de la agrupación de requisitos 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 gestionar.js y gestionar.phtml conforman la capa arquitectónica de presentación. La clase GestionarController sólo maneja la comunicación entre la vista y el modelo, la clase DatMessagebusModel es la encargada de la lógica del negocio, implementando funcionalidades que garantizan el cumplimiento de los requisitos identificados.

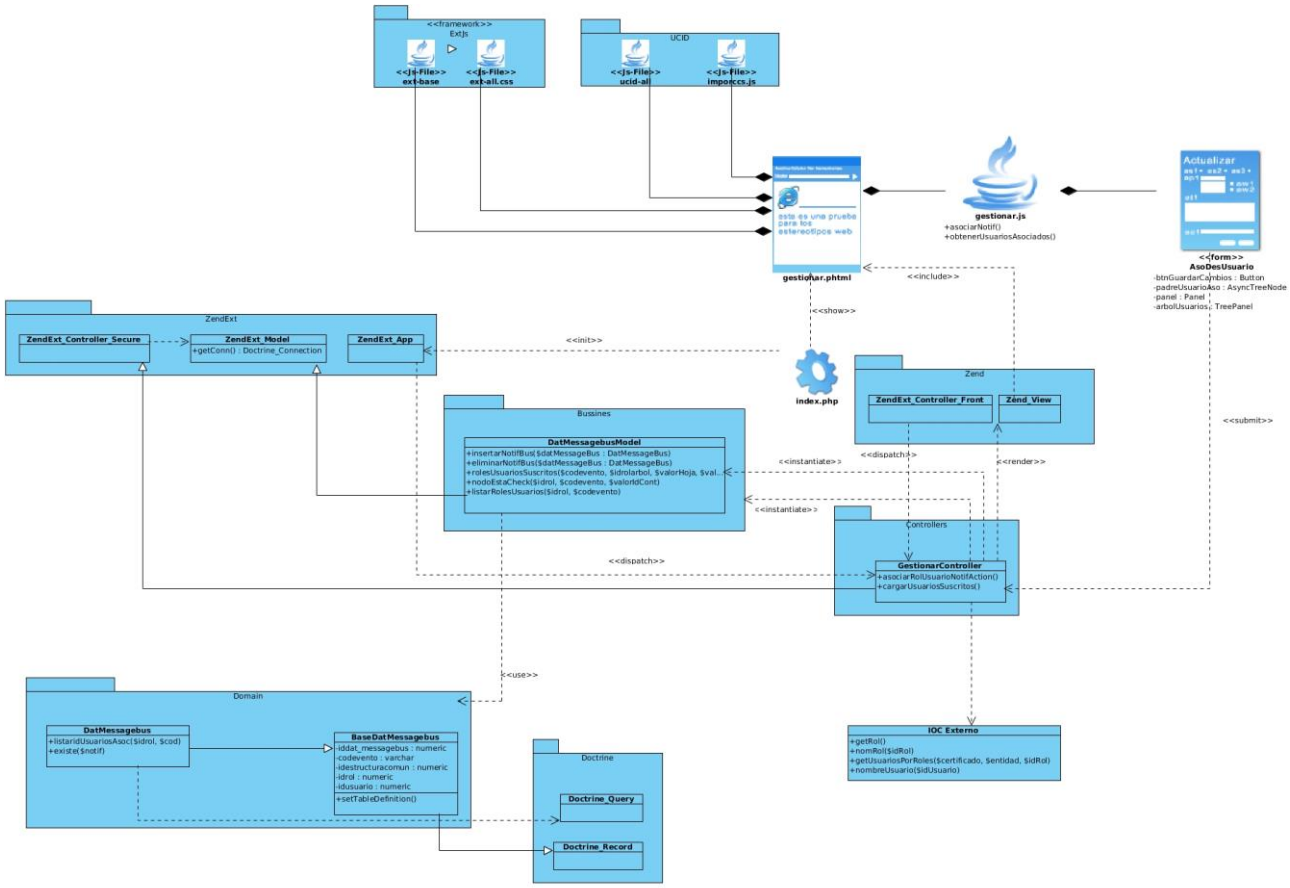


Figura 6: Diagrama de clases del diseño Gestionar Usuarios

2.4.5 Diagramas de secuencia

Los diagramas de secuencia modelan el flujo de la lógica dentro del sistema de forma visual, permitiendo documentarla y validarla. Proporcionan una buena base para identificar el

comportamiento del sistema. Son una buena herramienta para explorar la lógica de una operación compleja o los elementos implicados en la prestación de un servicio (Díaz, 2014). Las figuras 7, 8 y 9 muestran el diagrama de secuencia del requisito funcional Asociar usuario a notificaciones.

El flujo comienza cuando el usuario selecciona una notificación y seguidamente despliega el árbol de roles que muestra el formulario. El formulario envía la información a la clase controladora y esta pide al fichero subsemitennotif.xml todos los roles y usuarios suscritos. La clase DatMessagebusModel pide la información de los roles a través del servicio getRol (), haciendo uso del IOC. Se comprueba si el nodo del árbol está marcado para entonces solicitar la lista de usuarios por roles. A través del servicio nombreUsuario () que se consume haciendo uso del IOC se obtiene el nombre de todos los usuarios del sistema. Una vez mostrado el listado de los roles del sistema al usuario, este despliega el rol deseado y se muestra el listado de los usuarios que pertenecen al mismo. El usuario selecciona el rol o los usuarios a asociar y presiona el botón Guardar cambios. El formulario envía la información a la clase controladora y esta la recibe a través del método asociarRolUsuarioNotifAction. La información se almacena en la base de datos y se le muestra al usuario el mensaje: “Se han guardado los cambios satisfactoriamente”.

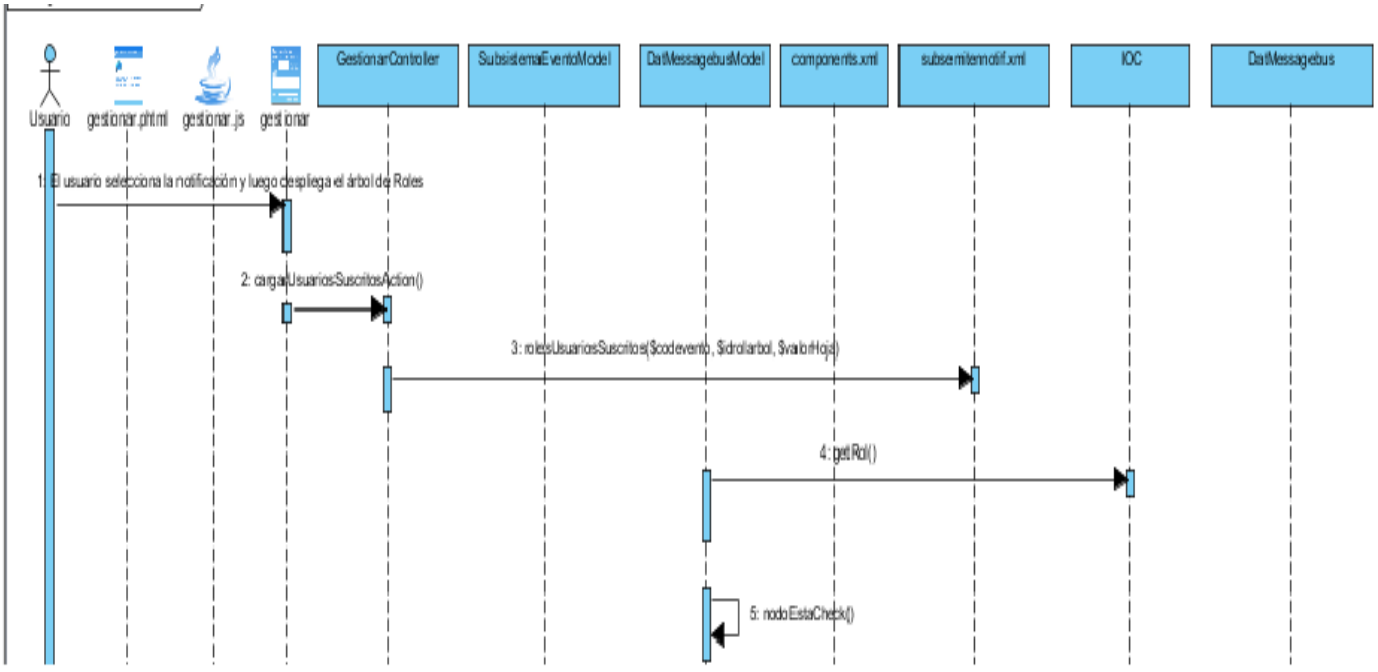


Figura 7: Fragmento 1 del diagrama de secuencia Asociar usuario a notificaciones

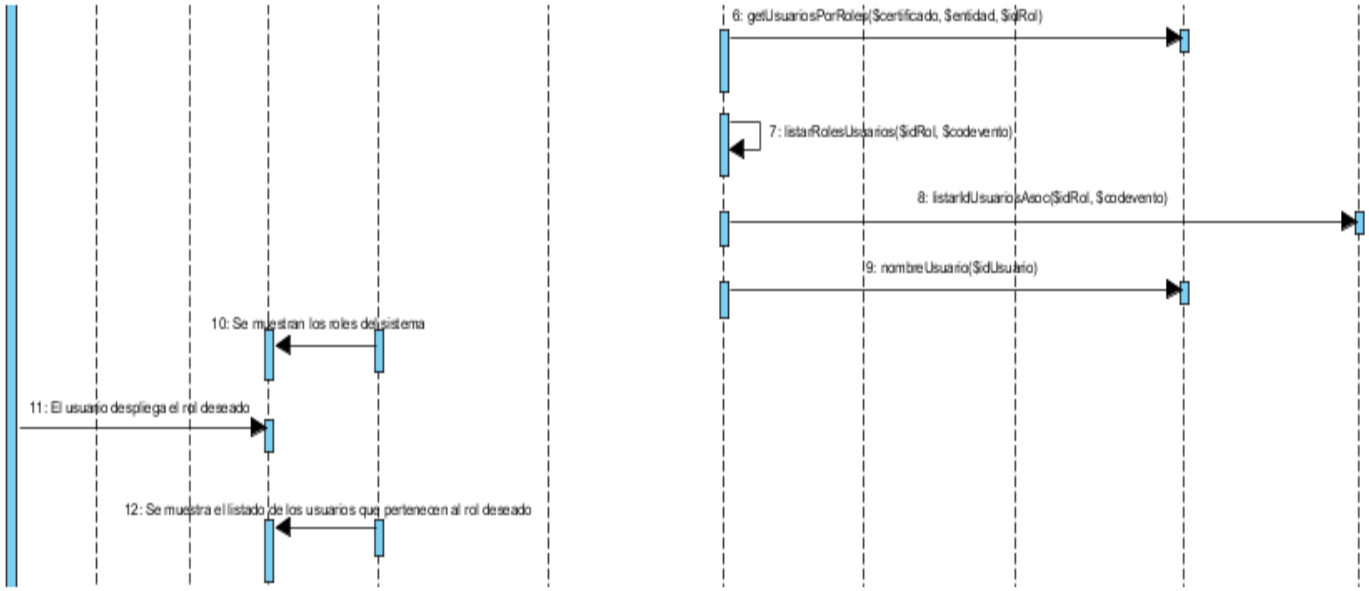


Figura 8: Fragmento 2 del diagrama de secuencia Asociar usuario a notificaciones

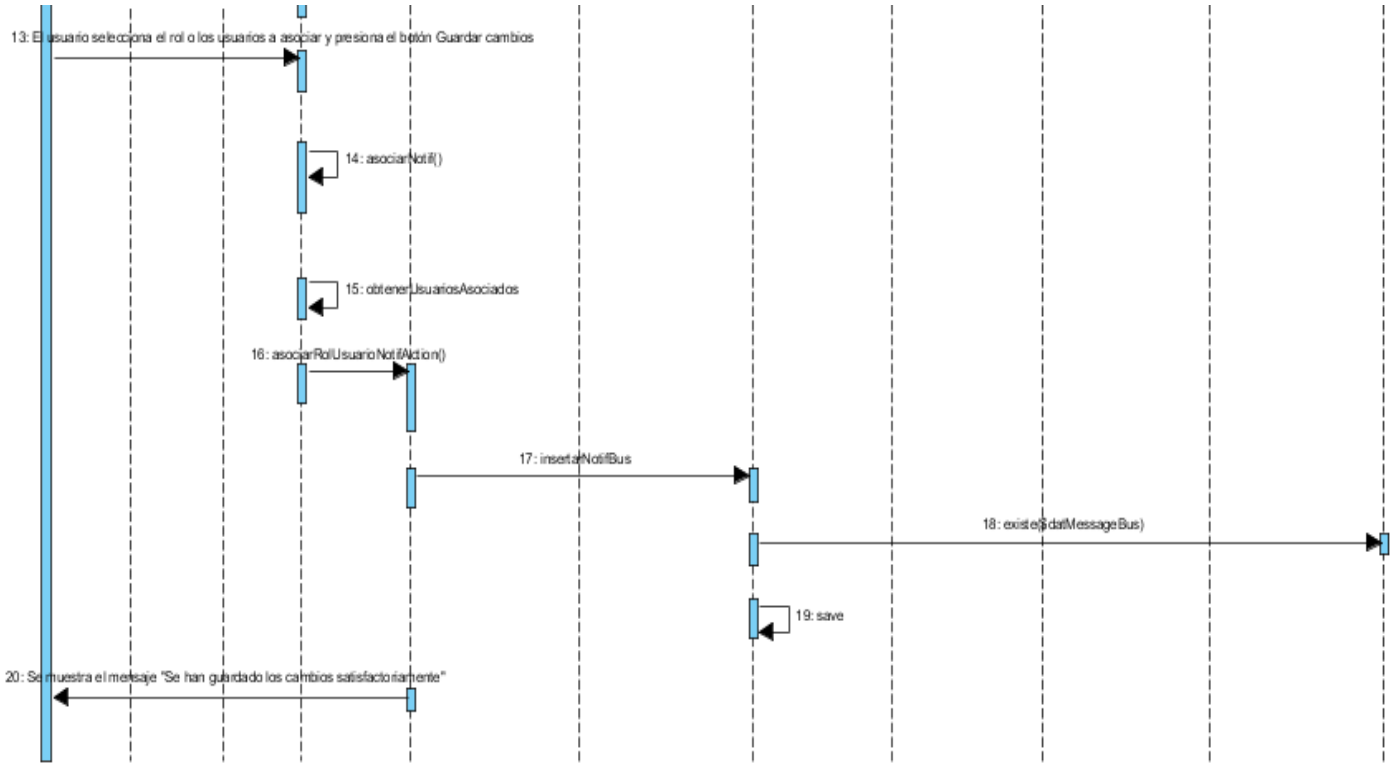


Figura 9: Fragmento 3 del diagrama de secuencia Asociar usuario a notificaciones

2.4.6 Modelo de datos

Puede definirse un modelo de datos como un conjunto de conceptos, reglas y convenciones que permiten construir una representación organizada de un sistema real (Lorenzo Castro, 1999-2000). Para el desarrollo de las mejoras funcionales al componente de gestión de notificaciones, se establecieron los conceptos y las relaciones que existen entre ellos.

En la Figura 8 se presenta el modelo de datos correspondiente al componente, al cual se le han adicionado las entidades ***dat_regla_notif***, ***nom_regla*** y ***dat_evento*** propias del negocio en cuestión. En el mismo se evidencia la utilización de los patrones llave subrogada en cada una de las entidades, mientras que entre las entidades ***nom_regla***, ***dat_regla_notif*** y ***dat_messagebus*** se evidencia un patrón jerárquico constituyendo un árbol fuertemente codificado. En la entidad ***dat_messagebus*** se ha adicionado la columna ***idusuario***, la cual permite establecer una relación entre los usuarios y las notificaciones.

El modelo de datos se encuentra en tercera forma normal ya que los atributos repetidos deben eliminarse y colocarse en tablas separadas; todas las dependencias parciales se deben eliminar y separar dentro de sus propias tablas; y todas las columnas que no son llaves, deben ser funcionalmente dependientes por completo de la llave primaria y no debe haber dependencias transitivas.

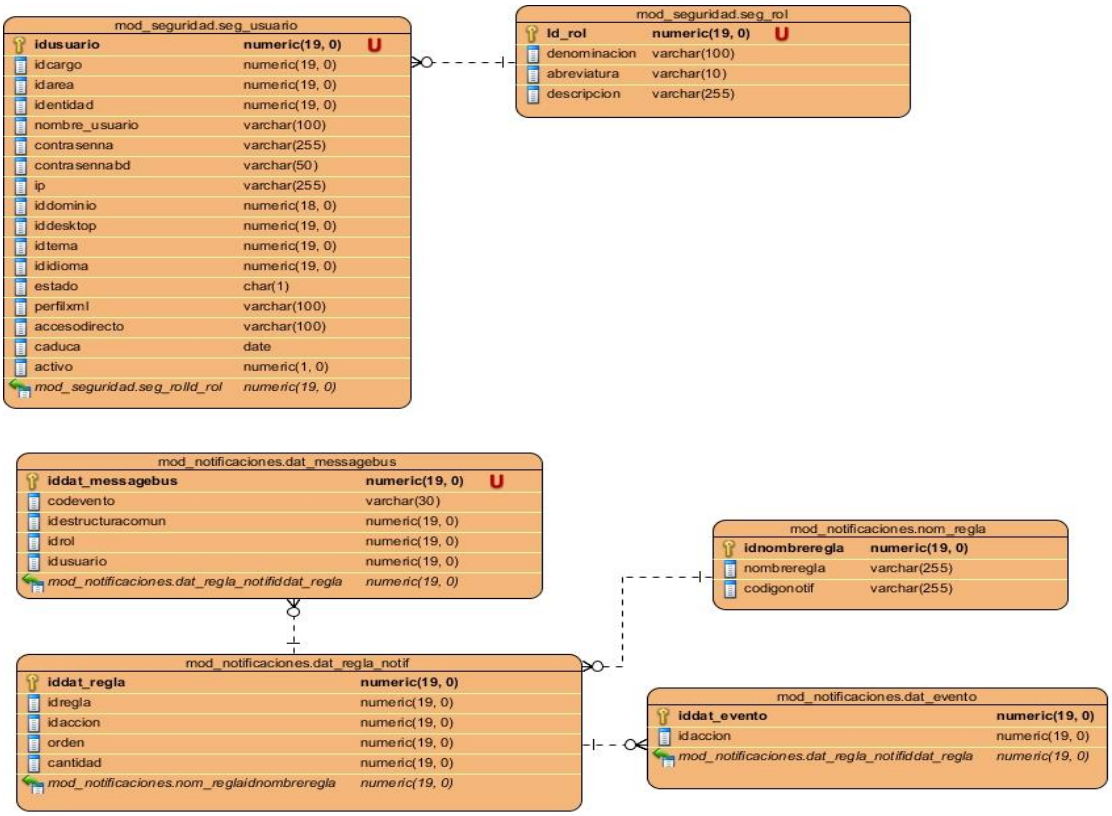


Figura 10: Modelo de datos

2.5Validación del diseño

Un aspecto importante a tener en cuenta en la evaluación del diseño, es la creación de métricas básicas inspiradas en el estudio de la calidad del diseño orientado a objetos. Para validar el diseño realizado se tendrán en cuenta distintos atributos de calidad como: la responsabilidad de las clases, la reutilización, la complejidad de implementación y mantenimiento, el bajo acoplamiento y la cantidad de pruebas unitarias. Las métricas Tamaño Operacional de Clases (TOC) y Relaciones entre Clases (RC) permiten evaluar estos atributos (Pressman, 2005).

Tamaño Operacional de Clase (TOC): está dado por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad (Lorenz, y otros).

Tabla 4: Atributos de calidad evaluados por la métrica TOC (Lorenz, y otros)

Atributo de calidad	Modo en que lo afecta
Responsabilidad	Aumento del TOC provoca aumento de la responsabilidad asignada a la clase.

Complejidad de implementación	Aumento del TOC provoca aumento de la complejidad de implementación de la clase.
Reutilización	Aumento del TOC provoca disminución del grado de reutilización de la clase.

Los criterios y categorías definidos para la evaluación de los atributos de calidad anteriores se presentan en la siguiente tabla:

Tabla 5: Criterios de evaluación para la métrica TOC (Lorenz, y otros)

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

Relaciones entre Clases (RC): está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad: (Lorenz, y otros)

Tabla 6: Atributos de calidad evaluados por la métrica RC (Lorenz, y otros)

Atributo de calidad	Modo en que lo afecta
Acoplamiento	Aumento del RC provoca aumento del Acoplamiento de la clase.
Complejidad de mantenimiento	Aumento del RC provoca aumento de la complejidad del mantenimiento de la clase.
Reutilización	Aumento del RC provoca disminución en el grado de reutilización de la clase.

Cantidad de pruebas	Aumento del RC provoca aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Los criterios y categorías definidos para la evaluación de los atributos de calidad anteriores se presentan en la siguiente tabla:

Tabla 7: Criterios de evaluación para la métrica RC (Lorenz, y otros)

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
Complejidad de 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

Resultados obtenidos en la aplicación de la métrica TOC

Luego de aplicarse la métrica de diseño TOC se obtuvieron resultados que permiten evaluar el diseño propuesto de calidad aceptable teniendo en cuenta que el 57% de las clases empleadas en el sistema poseen de 1 a 5 operaciones y solamente el 43% implementan entre 6 y 8 funcionalidades, lo que conlleva a evaluaciones positivas de los atributos de calidad involucrados (responsabilidad, complejidad de implementación y reutilización).

Figura 11: Resultados de la evaluación de la métrica TOC para el atributo Responsabilidad.

Figura 12: Resultados de la evaluación de la métrica TOC para el atributo Complejidad.

Figura 13: Resultados de la evaluación de la métrica TOC para el atributo Reutilización.

Resultados obtenidos de la aplicación de la métrica RC

Luego de aplicarse la métrica de diseño RC se obtuvieron resultados que permiten evaluar el

diseño propuesto con calidad aceptable, teniendo en cuenta que el 29% de las clases empleadas en el sistema no poseen dependencias y el 71% solamente tienen una dependencia con otra clase. Esto conlleva a evaluaciones positivas de los atributos de calidad involucrados (acoplamiento, complejidad de mantenimiento, cantidad de pruebas y reutilización).

Figura 14: Resultados de la evaluación de la métrica RC para el atributo Acoplamiento.

Figura 15: Resultados de la evaluación de la métrica RC para el atributo Complejidad de mantenimiento.

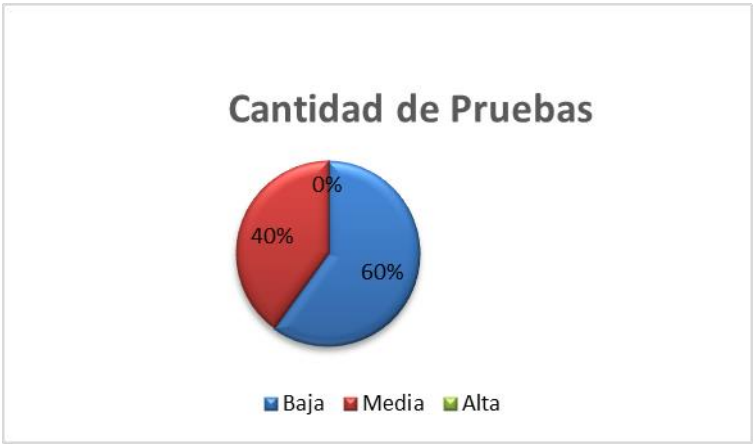


Figura 16: Resultados de la evaluación de la métrica RC para el atributo Reutilización.

Figura 17: Resultados de la evaluación de la métrica RC para el atributo Cantidad de pruebas.

2.6 Conclusiones parciales

En este capítulo se definieron los requisitos funcionales del sistema y las diferentes técnicas empleadas para la captura y validación de los mismos. Se desarrolló el modelo de diseño obteniendo como resultado una serie de productos de trabajo y se validó el mismo desde el punto de vista técnico, utilizando las métricas TOC y RC.

Capítulo III: IMPLEMENTACIÓN, PRUEBAS Y VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

3.1 Introducción

En el presente capítulo se continúa con el proceso de validación de requisitos mediante la implementación, la realización de pruebas y las validaciones. Se documentan los estándares de codificación utilizados, así como la descripción de los elementos significativos del código y el funcionamiento del componente de gestión de notificaciones. Además se evidencian las pruebas aplicadas a la solución y finalmente se validan las variables presentes en la investigación.

3.2 Implementación

Tomando como punto de partida los productos de trabajo generados en el análisis y diseño, se generan los ficheros de código que respondan a las funcionalidades del sistema solicitadas mediante la lógica de negocio diseñada (CEIGE, 2013).

A continuación se tratan los aspectos fundamentales que se realizaron durante esta etapa comenzando con el diagrama de componentes.

3.2.1 Diagrama de componentes

El conjunto de componentes y la representación de sus relaciones de manera gráfica constituyen un diagrama de componentes (Ferré Grau, y otros). Definiéndose componente como la representación física de una parte de un sistema modular y reemplazable que encapsula la implementación (Object Management Group, 2001).

En la Figura 16 se ilustra el diagrama de componentes correspondiente a la propuesta de solución, en el cual el componente Seguridad a través de la interfaz SeguridadProxyService provee al componente Notificaciones de servicios para la obtención de roles, usuarios, usuarios por roles y nomenclador de roles. A su vez el componente ZendExt consume el servicio para la ejecución de reglas de negocio brindado por el componente Notificaciones. Para instalar el componente Notificaciones en otros sistemas es necesario tener en cuenta al componente Estructura y Composición, pues el mismo brinda el servicio para la obtención de estructuras.

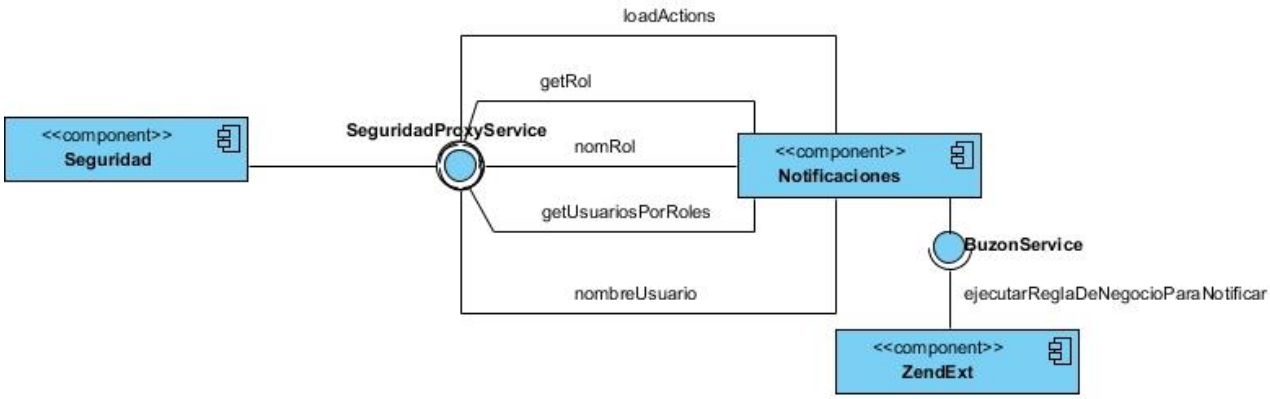


Figura 18: Diagrama de componentes

3.2.2 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. Establecerán las pautas que conlleven a lograr un código más legible y reutilizable, de tal forma que pueda aumentar su mantenibilidad a lo largo del tiempo (CEIGE, 2012).

Seguidamente se especifican los estándares de codificación usados para el desarrollo de la solución:

- **Notación PascalCasing:** 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.

Nomenclatura de las clases

Los nombres de las clases comienzan con la primera letra en mayúscula y el resto en minúscula.

- En las clases controladoras los nombres irán seguidos de la palabra: “Controller”. Ejemplo: GestionarController.
- Las clases de los modelos que se encuentran dentro de la carpeta del negocio después del nombre llevan la palabra: “Modelo”. Ejemplo: DatMessagebusModel.
- Las clases que se encuentran dentro de la carpeta del dominio el nombre que reciben es el de la tabla en la Base de Datos. Ejemplo: DatMessagebus.
- Las clases que se encuentran dentro de la carpeta generado comienzan con la palabra:

"Base" y seguido el nombre de la tabla en la Base de Datos. Ejemplo: BaseDatMessagebus.

- En la vista los nombres comienzan con las letras iniciales mayúsculas haciendo referencia a la agrupación de requisitos que se maneja. La agrupación de requisitos adicionar, modificar, eliminar y listar, se le pone AMEL_Nombre de la clase. Ejemplo: AMELReglaDeNegocio.

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 a partir de la segunda palabra comienzan con mayúscula. En la clase controladora van seguidas de la palabra "Action". Ejemplo: cargarUsuariosSuscritosAction().

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 a partir de la segunda palabra letra inicial mayúscula. Además, debe iniciar con el tipo de dato. Ejemplo: $ idUsuario.

Una vez establecidas las bases para el desarrollo de la solución, se lleva a cabo la implementación de las funcionalidades identificadas. Se describen a continuación los elementos significativos del código y el funcionamiento del componente.

3.2.3 Descripción de la implementación

Para desarrollar la extensión del componente de gestión de notificaciones del marco de trabajo Sauxe se hizo necesario implementar una serie de funcionalidades. La tabla 8 explica los servicios que fueron consumidos y brindados por el componente.

Dando respuesta al requisito funcional Gestionar notificaciones, se modificó la funcionalidad **Configurar notificaciones** del componente, para que los avisos llegaran no sólo a los roles del sistema suscritos a alguna notificación, sino también a los usuarios asociados a ese rol y que podían ser suscriptores de alguna notificación. Fue necesario implementar una serie de servicios y métodos con el objetivo de obtener la información necesaria para poder asociar o desasociar a los usuarios de las notificaciones.

Para mostrar el listado de usuarios perteneciente a un rol, fue necesario implementar la función listarRolesUsuarios, quien recibe como parámetros el identificador del rol y el código de la notificación. Primeramente se crea un objeto de la clase DatMessagebus para tener acceso a los

métodos de la misma. Luego, se instancia la función listaridUsuariosAsoc y teniendo en cuenta los parámetros iniciales, busca en la base de datos qué usuarios cumplen con ellos y los imprime.

```
118     public function listarRolesUsuarios($idrol, $codevento)
119     {
120         $mb = new DatMessagebus();
121         $usuarios = $mb->listaridUsuariosAsoc($idrol, $codevento);
122         return $usuarios;
123     }
```

Figura 19: Código del método listarRolesUsuarios

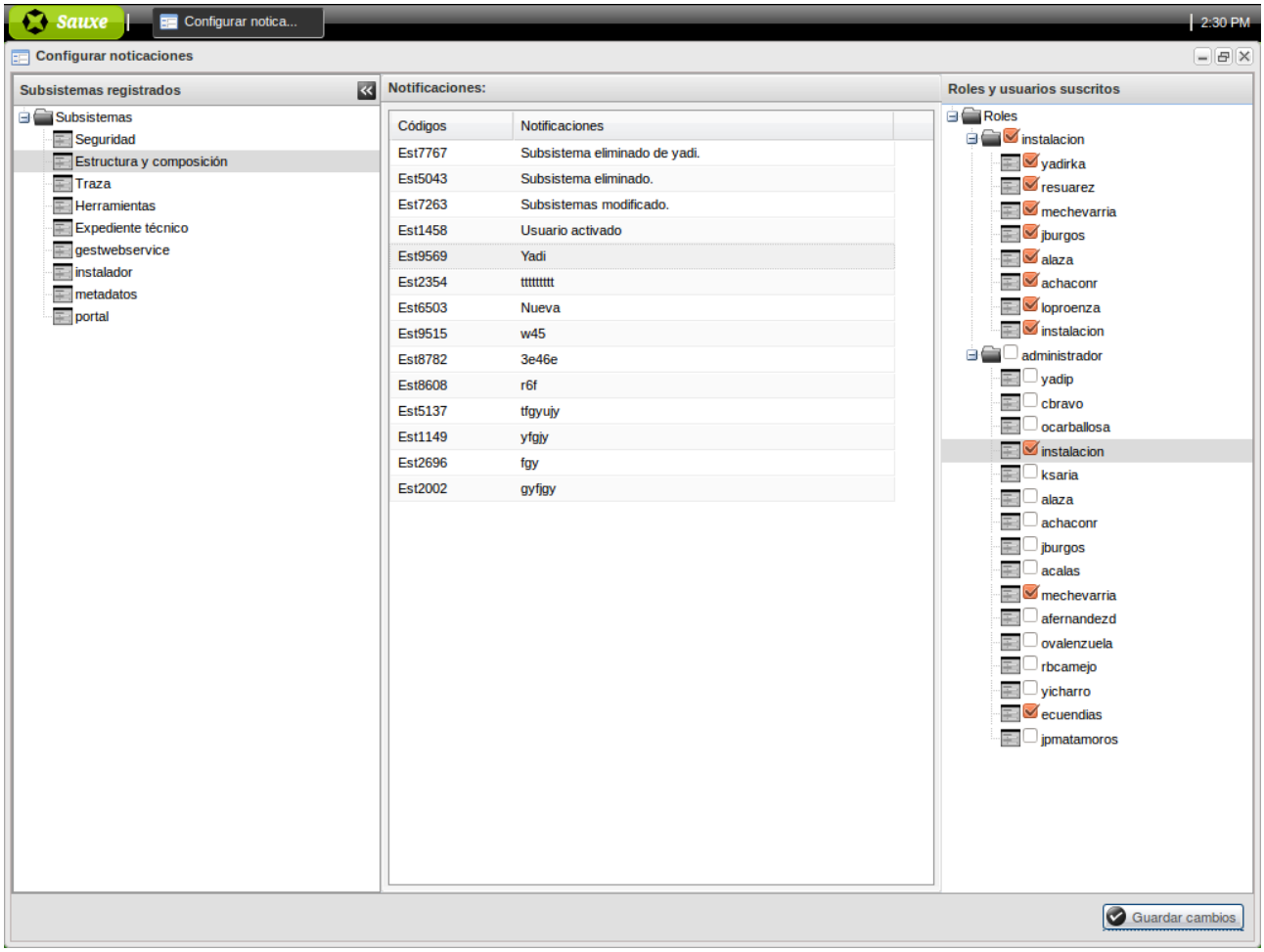


Figura 20: Interfaz para asociar o desasociar usuario a notificaciones

También fue necesario añadir al componente, la funcionalidad **Gestionar reglas de negocio** con el objetivo de dar solución al requisito funcional de igual nombre y de que el proceso de notificar se haga de la forma más simple y dinámica posible. Sin embargo, el gestor de reglas de negocio del componente de gestión de notificaciones se ve restringido debido a que no hay forma de saber todas las acciones que existen en las controladoras del sistema, a menos que dichas acciones se hayan registrado en el sistema usando la opción Acciones, del componente Configurar sistemas, perteneciente al subsistema Seguridad.

Para capturar los eventos que ocurren dentro del marco de trabajo y poder compararlos con los definidos en el gestor de reglas de negocio, se implementa el servicio "***ejecutarReglaDeNegocioParaNotificar***" en la clase "***GestionarReglaNegocioModel***" y se invoca en el método postDispatch de la clase ZendExt_Controller_Secure.

Una vez ejecutada una acción, se captura el identificador de la misma para buscar las reglas a la que pertenece. Si existe alguna coincidencia, se observa el orden de las acciones de la regla y la cantidad de veces que estas se han ejecutado y se almacenan en la tabla ***dat_evento***. Una vez cumplidos el orden y el número de veces de ejecución de los eventos, se genera la notificación y se borran las acciones para reiniciar el contador de eventos.

```
137     public function ejecutarReglaDeNegocioParaNotificar($idaccion)
138     {
139         $datRegla = new DatReglaNotif();
140 //buscar las reglas donde pertenece la accion
141         $aux = $datRegla->listarReglaDadoIdAccion($idaccion);
142         $arrayIdReglas = $this->elimina_duplicados($aux, 'idregla');
143         if ($arrayIdReglas != NULL) {
144 //si hay reglas definidas buscar el orden de las acciones de la regla
145             foreach ($arrayIdReglas as $unIdRegla) {
146                 $arrayOrden = array();
147                 $arrayOrden = $datRegla->listarAccionesRegla(10000000, 0, $unIdRegla['idregla']);
148 
149 //Si cumple con la regla la adiciono en DatEvento
150                 $this->adicionarAccionEvento($idaccion, $arrayOrden);//no esta adicionando
151 
152 //si se cumple el orden de toda la regla notifico
153                 if ($this->comprobarOrden($arrayOrden)) {
154 
155                     $estaRegla = NomReglaNotif::buscarTipoRegla($arrayOrden[0]['idnombreregla']);
156                     $nombreRegla = $estaRegla[0]['nombreregla'];
157                     $codigoNotif = $estaRegla[0]['codigonotif'];
158                     $moduloNotif = SubsistemaeventoModel::subsistemaEmiteNotif($codigoNotif);
159 
160                     $test2['mensaje'] = 'Se ejecuto la regla: ' . $nombreRegla . ' .';
161                     $messagebus = ZendExt_MessageBus::getInstance();
162                     $messagebus->FireEvent($moduloNotif['subsistema'], $codigoNotif, $test2);
163 //elimino las acciones despues de notficado para reiniciar contador
164                     $this->eliminarAccionesEvento($arrayOrden);
165                 }   }
166             }   }
```

Figura 21: Código del servicio ejecutarReglaDeNegocioParaNotificar

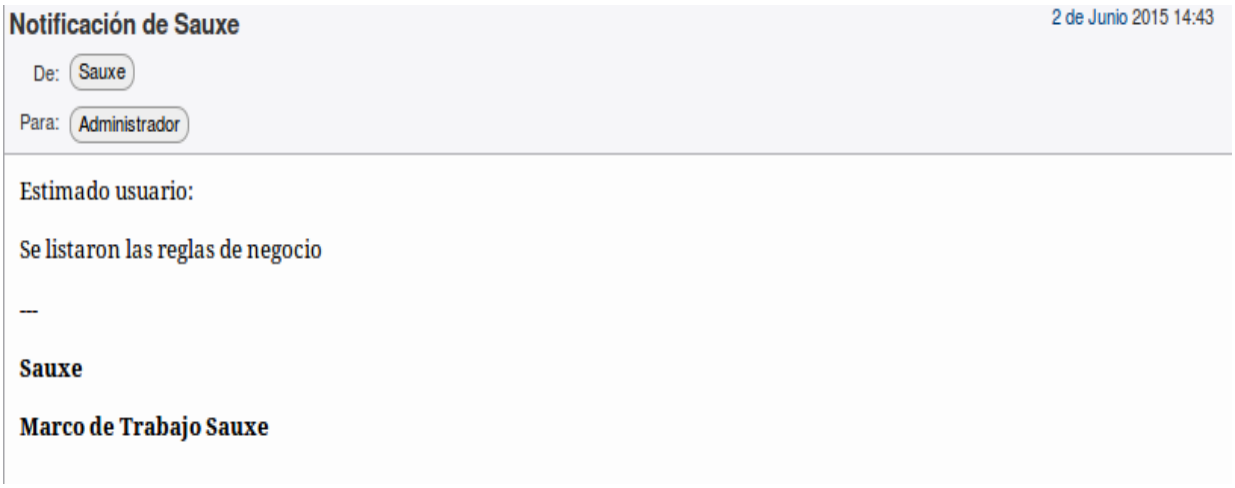


Figura 22: Notificaciones en el correo

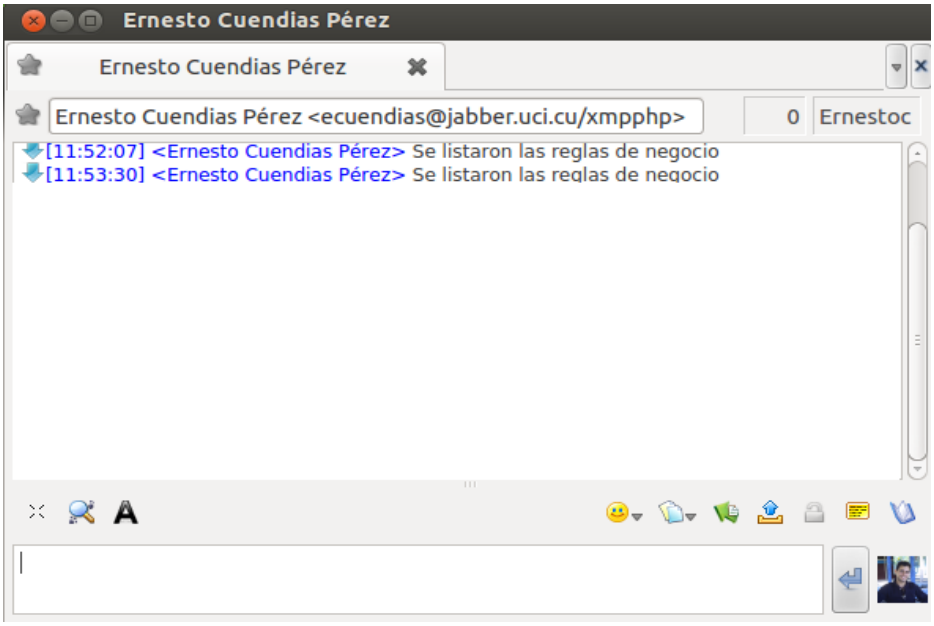


Figura 23: Notificaciones en el jabber

Tabla 8: Servicios que brinda y consume el componente Notificaciones

Servicio	Brindado por	Consumido por	Parámetros	Descripción
getRol	Seguridad	Notificaciones	-	Obtiene los roles registrados en el sistema.
nomRol	Seguridad	Notificaciones	idRol	Devuelve el nombre de los roles registrados en el sistema.
getUsuariosPorRoles	Seguridad	Notificaciones	certificado	Obtiene los usuarios del

			entidad idRol	sistema por el rol al que pertenecen.
nombreUsuario	Seguridad	Notificaciones	idUsuario	Devuelve el nombre de los usuarios registrados en el sistema
loadActions	Seguridad	Notificaciones	idSubsistema, idaccion	Devuelve un listado con todos los eventos que ocurren en el sistema.
ejecutarReglaDeNegocioParaNotificar	Notificaciones	ZendExt_Controller_Secure	idAccion	Ejecuta la regla de negocio asociada al evento que ha ocurrido.

3.3 Pruebas de software

Las pruebas de software sirven para evaluar la calidad de un producto de software o para mejorarlo, mediante la identificación de sus defectos y problemas. Consiste en la verificación dinámica del comportamiento real de un programa frente al comportamiento esperado, para un conjunto finito de casos de prueba (convenientemente seleccionados entre las usualmente infinitas posibilidades de ejecución) (Ruiz, 2009).

Las pruebas tienen gran importancia en el desarrollo de un sistema informático, ya que mediante éstas se pueden detectar y corregir posibles fallos de implementación, calidad o usabilidad tempranamente y son un elemento crítico para la garantía del correcto funcionamiento del software.

La práctica de pruebas en la construcción de software reduce el tiempo de desarrollo reduciendo la cantidad de tiempo necesario para integrar y estabilizar versiones. Mejora la productividad encontrando y arreglando errores rápidamente, además de incrementar la calidad global del software garantizando que todo el código nuevo ha sido probado, y a todo el código existente se le aplican pruebas de regresión antes de ser integrados a la base central de código.

3.3.1 Prueba de Caja Blanca

Las pruebas de caja blanca realizan un seguimiento del código fuente según se van ejecutando los casos de prueba, de manera que se determinan de forma concreta las instrucciones en las que existen errores. Requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas del código.

Objetivo

El objetivo de realizar este tipo de prueba al sistema es garantizar que se ejerciten al menos una

vez todos los caminos independientes de cada módulo o método, todos los bucles en sus límites operacionales, las estructuras internas de datos y que se alcancen todas las decisiones lógicas en sus vertientes verdadera y falsas para asegurar su validez (Pressman, 2005).

Alcance

Entre las técnicas para obtener casos de prueba de caja blanca se encuentra la del camino básico. Este método permite obtener una medida de la complejidad lógica de un diseño procedimental que posibilita definir un conjunto básico de caminos de ejecución. Para la obtención de la complejidad ciclomática se emplea una representación del flujo de control en forma de grafo (Departamento de Informática, 2007).

La complejidad ciclomática define el número de caminos independientes de un programa. Por camino independiente se entiende aquel que introduce una arista que no haya sido recorrida antes. Da una cota o límite superior para el número de casos de prueba necesarios para ejercitar cada sentencia del código (Departamento de Informática, 2007).

Descripción

La prueba de caja blanca es considerada como uno de los tipos de pruebas más importantes que se le aplican al software, logrando como resultado que disminuyan en gran porciento el número de errores existentes en los sistemas, aumentando la calidad y confiabilidad de los mismos (Pressman, 2005). Para el desarrollo de estas pruebas se utilizó la técnica camino básico.

Desarrollo del método

Se cuenta con el requisito Asociar usuario a notificaciones para configurar las notificaciones hasta el nivel de usuarios. A continuación se muestra el camino del método asociarRolUsuarioNotifAction(), el cual es parte de la ejecución de este requisito. La figura 22 muestra el fragmento de código correspondiente al método:

```php
public function asociarRolUsuarioNotifAction()
{
    $useRolSuscribed = $this->_request->getPost('asociados');   //1
    $codUseRolSusc = json_decode(stripslashes($useRolSuscribed));
    $databusModel = new DatMessagebusModel();


    foreach ($codUseRolSusc as $uno) {  //2

        $databus = new DatMessagebus();  //3
        $databus->idestructuracomun = $this->global->Estructura->idestructura;
        $databus->idrol = $uno->idRolEnvio;
        $databus->idusuario = $uno->iduserEnvio;
        $databus->codevento = $uno->codigoEnvio;

        if ($uno->checkedEnvio == 1) {   //4
            $databusModel->insertarNotifBus($databus);  //5
        } else {
            $databusModel->eliminarNotifBus($databus);  //6
        }
    }  //7
    echo json_encode(array('mensaje' => 'Se han guardado los cambios satisfactoriamente.', 'codMsg' => 1));  //8
}
```

Figura 24: Código del método asociarRolUsuarioNotifAction

Dado el código, se hace el grafo que se muestra en la siguiente figura:

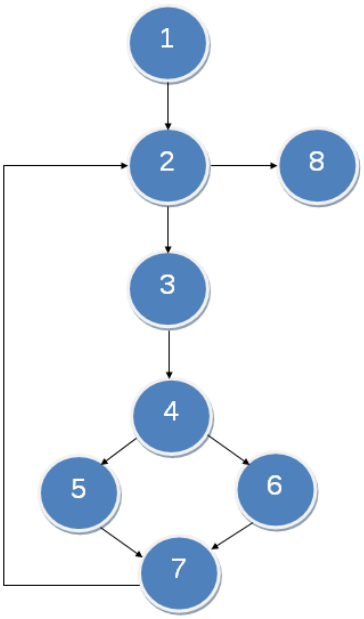


Figura 25: Grafo del método asociarRolUsuarioNotifAction

Se calcula la complejidad ciclomática utilizando las fórmulas que se muestran a continuación, siendo A la cantidad de aristas, N la cantidad de nodos, P la cantidad de nodos predicados y R la

cantidad de regiones contenidas en el grafo.

V(G)=A-N+2, V(G)= 9-8+2, V(G)=3

V(G)=P+1, V(G)=2+1, V(G)=3

V(G)=R, V(G)=3

Para cada una de ellas el resultado es de 3, determinando la cantidad de casos de prueba.

Camino 1: 1-2-8

Camino 2: 1-2-3-4-5-7-2-8

Camino 3: 1-2-3-4-6-7-2-8

Caso de prueba para el camino básico # 1

Camino: 1-2-8

Descripción: El parámetro $asociados está vacío.

Entrada: $asociados = [ ].

Resultados esperados: No se asocia ni desasocia ningún usuario.

Resultados obtenidos: Satisfactorio.

Caso de prueba para el camino básico # 2

Camino: 1-2-3-4-5-7-2-8

Descripción: Los datos de entrada cumplirán con los siguientes requisitos:

El parámetro $asociados no puede estar vacío.

Entrada: $asociados = [{ idRolEnvio = 220000000019,
idUserEnvio = 220000000037,
codigoEnvio = Est9569,
checkedEnvio = TRUE }].

Resultados esperados: Se asoció el usuario a la notificación.

Resultados obtenidos: Satisfactorio.

Caso de prueba para el camino básico # 3

Camino: 1-2-3-4-6-7-2-8

Descripción: Los datos de entrada cumplirán con los siguientes requisitos:

El parámetro $asociados no puede estar vacío.

Entrada: $asociados = [{ idRolEnvio = 220000000019,
idUserEnvio = 220000000037,
codigoEnvio = Est9569,
checkedEnvio = FALSE }].

Resultados esperados: Se desasoció el usuario de la notificación.

Resultados obtenidos: Satisfactorio.

3.3.2 Prueba de Caja Negra

Las pruebas de caja negra se llevan a cabo sobre la interfaz de usuario, se centran principalmente en los requisitos funcionales del software. Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Examinan aspectos del modelo, principalmente del sistema, sin tener en cuenta la estructura interna del software.

Objetivo

El objetivo de realizar pruebas de caja negra a un sistema es el de detectar fallos que impidan el correcto o completo funcionamiento de este, así como los errores de interfaces, rendimiento, de inicialización y terminación (Pressman, 2005).

Alcance

El proceso de pruebas de caja negra se va a centrar principalmente en los requisitos funcionales del software para verificar el comportamiento de la unidad observable externamente y su calidad, por lo que se llevarán a cabo las pruebas funcionales sobre las interfaces del sistema y la de aceptación, por parte del cliente.

Descripción

Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales del programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

La prueba de caja negra no es una alternativa a las técnicas de prueba de la caja blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la caja blanca (Grupo Alarcos, 2010).

Las pruebas se realizaron sobre todos los requisitos funcionales del componente, haciendo uso de los casos de prueba correspondientes a cada uno de estos. A continuación se muestra como ejemplo: el caso de prueba que responde al requisito Asociar usuario a notificaciones. El resto de los casos de prueba se encuentran en el expediente de proyecto.

Descripción general

El sistema debe permitir asociar al menos un usuario a alguna notificación

Condiciones de ejecución

- Se ha registrado al menos un usuario en el sistema.
- Se debe identificar y autenticar ante el sistema, además debe tener los permisos para ejecutar esta acción.

•Los subsistemas registrados deben tener al menos una notificación asociada

Asociar usuario a notificaciones

Escenario	Descripción	usuario	rol	Respuesta del sistema	Flujo central
EC 1.1: Asociar usuario a la notificación	Todos los usuarios del rol seleccionado se deben asociar a la notificación	V yadirka	V instalacion	El sistema muestra el mensaje: Se han guardado los cambios	- Se selecciona uno de los subsistemas
EC 1.2: Asociar usuario a la notificación seleccionada desmarcando al menos un usuario de	Todos los usuarios marcados se deben asociar a la notificación seleccionada.	V mechevarria	V instalacion	El sistema muestra el mensaje: Se han guardado los cambios satisfactoriamente	- Se selecciona uno de los subsistemas registrados. - Se selecciona alguna

Figura 26: Fragmento del Caso de Prueba Asociar usuario a notificaciones.

3.3.3 Resultado de las pruebas

Las pruebas realizadas al sistema fueron satisfactorias desde el punto de vista interno y funcional, estas abarcaron requerimientos, funciones y la lógica interna del software. Se aplicaron los métodos de caja negra y caja blanca para validar los requisitos.

Luego de resueltas las 9 NC detectadas durante las pruebas realizadas, el componente fue aprobado por la comisión de calidad interna del Departamento de Gestión de Componentes del CEIGE, encontrándose completamente disponible para su utilización.

Tabla 9: Total de NC detectadas en las pruebas de calidad interna

Componente	Iteración 1	Iteración 2	Iteración 3
Notificaciones	8	1	-

3.4 Validación de las variables

Con el desarrollo de las mejoras funcionales al componente de gestión de notificaciones del marco de trabajo Sauxe, se puede afirmar que las mismas han tenido un impacto positivo sobre las variables de esta investigación: usabilidad, interoperabilidad e independencia.

- **Usabilidad**

Antes de desarrollar el gestor de reglas de negocio, era muy complicado generar las notificaciones

pues había que acceder al código del componente interesado en ser notificado y escribir dentro las siguientes líneas:

```
$test2['mensaje'] = 'mensajeeee 555555';
$messagebus =  ZendExt_MessageBus::getInstance();
$messagebus->FireEvent('seguridad','Seg9251',$test2);
```

Esto implicaba que se pudieran cometer errores dentro del código y que el usuario necesariamente debía conocer sobre el funcionamiento del componente.

Con la implementación del gestor de reglas de negocio todo lo anterior fue eliminado ya que el proceso para generar una notificación se configura de manera gráfica, además se logró disminuir el tiempo para generar una notificación, pues de dos minutos que tomaba a un profesional, se redujo a 20 segundos.

- **Interoperabilidad e independencia**

Antes de añadir el gestor de reglas de negocio a Notificaciones, existía dependencia directa del código de los componentes por lo que era necesario implementar un servicio en cada uno para lanzar las alertas.

Luego de haberse desarrollado el gestor de reglas de negocio, esto desaparece ya que al configurarse todo el proceso de manera gráfica desde Notificaciones, disminuye considerablemente la dependencia entre este componente y el resto, lo que propicia un aumento de la interoperabilidad.

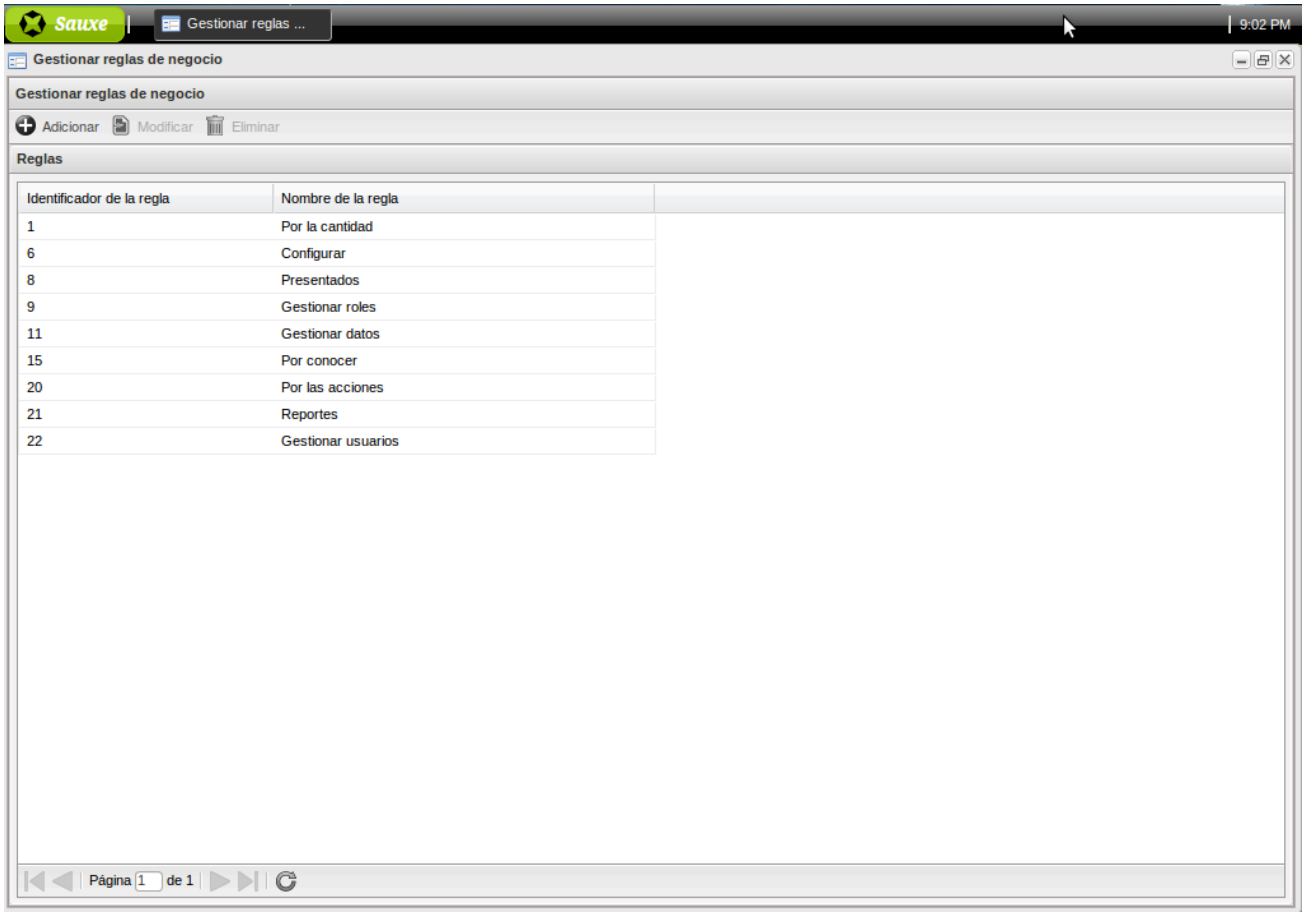


Figura 27: Interfaz de la funcionalidad Gestionar reglas de negocio

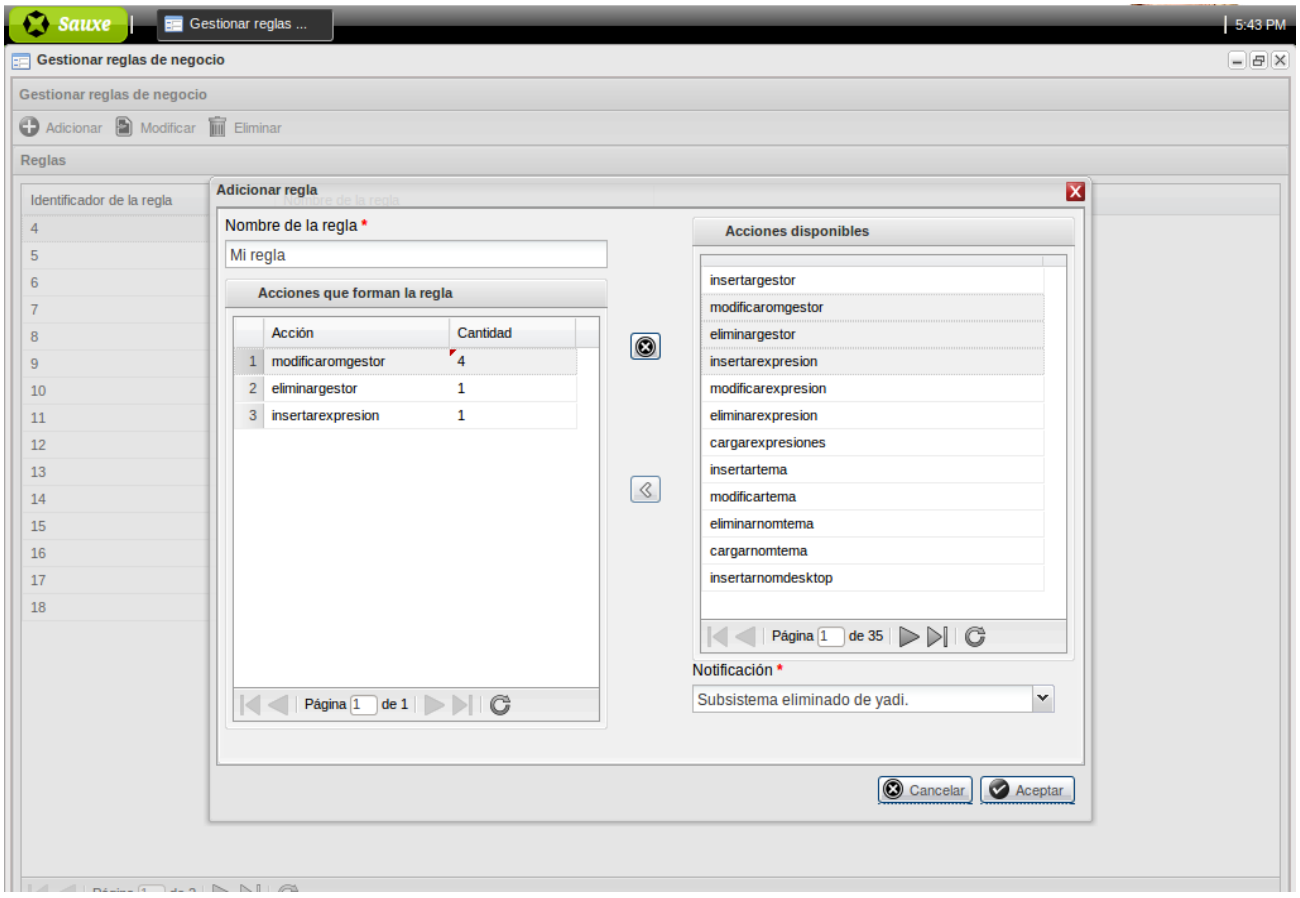


Figura 28: Interfaz de Adicionar reglas de negocio

3.5 Conclusiones parciales

Durante el transcurso del capítulo se extendió el componente con la implementación de las clases y funciones del diseño, se desarrollaron pruebas de caja blanca y caja negra en varias iteraciones que permiten validar la solución. Además se expusieron los escenarios de prueba para validar que las variables de la validación tomaron valores positivos, que permiten el cumplimiento de los objetivos del trabajo.

Conclusiones generales

Una vez culminado el trabajo y como resultado del mismo se logró alcanzar el objetivo general propuesto, tras cumplir todas las tareas trazadas a inicio de la investigación:

- Se realizó un estudio detallado de los sistemas que utilizan el envío de alertas, arrojando como resultado la necesidad de desarrollar una extensión del componente Notificaciones para el marco de trabajo Sauxe, que fuese capaz de enviar alertas mediante reglas de negocio hasta el nivel de usuario.
- Se realizó el análisis y diseño de las nuevas funcionalidades para extender el componente de Notificaciones del marco de trabajo Sauxe, para el envío de alertas mediante reglas de negocio hasta el nivel de usuario.
- Se logró extender el componente de notificaciones del marco de trabajo Sauxe para el envío de alertas mediante reglas de negocio a nivel de usuario, incrementando la usabilidad, la interoperabilidad y la independencia.
- Se aplicaron las pruebas de unidad, sistema y aceptación; validando tanto la interfaz de usuario como el adecuado funcionamiento interno del componente.

RECOMENDACIONES

Tomando como base la investigación realizada y la experiencia acumulada durante el desarrollo del presente trabajo de diploma se recomienda:

- Incluir al gestor de reglas de negocio un parámetro relacionado con el tiempo, el cual permita definir: minutos, hora, día, mes y año deseados para generar la notificación.
- Implementar una funcionalidad que, siguiendo la filosofía del buzón de notificaciones, permita almacenar las reglas de negocio que expiraron.
- Incluir al gestor de reglas de negocio una funcionalidad que permita filtrar o buscar las reglas adicionadas.

REFERENCIAS

1. **Barreto, Jorge Méndez. 2012.** *Plugin para garantizar la autenticidad de los usuarios de Openfire desde Acaxia.* La Habana : s.n., 2012.
2. **Baryolo Gómez, Oiner, Morejón Borbón, Yoandry y García Tejo, Darien. 2010.** *ARQUITECTURA TECNOLÓGICA PARA EL DESARROLLO DE SOFTWARE.* La Habana : s.n., 2010. s.n.
3. **Bizagi Limited. 2010.** Bizagi BPM Suite User's Guide. *Bizagi BPM Suite User's Guide.* [En línea] 2010. [Citado el: 21 de Junio de 2015.] http://help.bizagi.com/bpmsuite/es/index.html?definir_reglas_de_negocio.htm.
4. **CEIGE. 2012.** *Estándares de codificación Proyectos con el marco de trabajo Sauxe del CEIGE.* La Habana : s.n., 2012.
5. **—. 2013.** *Modelo de desarrollo de software.* La Habana : s.n., 2013.
6. **Colectivo de autores. 2010.** ¿Qué significa Evento? *¿Qué significa Evento?* [En línea] 2010. [Citado el: 8 de febrero de 2015.] http://www.definiciones-de.com/Definicion/de/evento.php.
7. **—. 2010.** Gestión de los requisitos. *Gestión de los requisitos.* [En línea] 2010. [Citado el: 26 de Marzo de 2015.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=RM#_Toc260742352..
8. **—. 2011.** *Manual de PHP.* La Habana : s.n, 2011.
9. **—. 2009.** The free dictionary - tiempo real. *The free dictionary - tiempo real.* [En línea] 2009. [Citado el: 6 de Febrero de 2015.] http://es.thefreedictionary.com/tiempo..
10. **CollabNet Inc. 2010.** Apache Subversion. *Apache Subversion.* [En línea] 2010. [Citado el: 18 de junio de 2015.] http://subversion.apache.org/.
11. **Departamento de Informática. 2007.** *Tema 7: Validación.* Universidad de Valladolid : s.n., 2007.
12. **Díaz, Ariel Trujillo. 2012.** *Componente para la gestión de notificaciones del marco de trabajo Sauxe.* La Habana : s.n., 2012.
13. **Díaz, Yadirka Verdecia. 2014.** *Desarrollo de mejoras funcionales de los módulos Taller y Recursos humanos de Órbita.* La Habana : s.n., 2014.
14. **doctrine.org. 2011.** Doctrine. *Doctrine.* [En línea] 2011. [Citado el: 6 de Febrero de 2015.] http://www.phpdoctrine.org..
15. **Escalona , José María y Koch, Nora. 2002.** *Ingeniería de Requisitos en Aplicaciones para la Web – Un estudio comparativo.* Sevilla, España : s.n., 2002. s.n.

16. **Escribano, Gerardo Fernández. 2002.** *Introducción a Extreme Programing.* 2002.
17. **Estanque Díaz, Diuver, Hidalgo López, Leydis y González Martínez, Yoandy. 2013.** *Diseño del mecanismo para alertas y notificaciones del sistema de información hospitalaria ALAS HIS.* La Habana : s.n., 2013. s.n.
18. **Ferré Grau, Xavier y Sánchez Segura, María Isabel.** *Desarrollo Orientado a Objetos con UML.* Facultad de Informática – UPM : s.n.
19. **Flanagan, David. 2006.** *JavaScript: The Definitive Guide.* s.l. : Anaya Multimedia, 2006. 978-84-415-2202-2 84-415-2202-2.
20. **Grupo Alarcos. 2010.** Alarcos. [En línea] 2010. [Citado el: 17 de mayo de 2015.] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema09.pdf.
21. **Ignite REALTIME. 2008.** Openfire Server. *Openfire Server.* [En línea] 2008. http://www.igniterealtime, org/projects/openfire.
22. **Larman, C. 2003.** *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* Madrid : Pearson Educación S.A, 2003.
23. **Lopez, Felipe. 2013.** Tutorial de PostgreSQL, 9.1.0. *Tutorial de PostgreSQL, 9.1.0.* [En línea] 2013. [Citado el: 7 de Febrero de 2015.]
24. **Lorente, Lianet Rodríguez. 2013.** *Solución para Alertas y Avisos en el Sistema de Planificación de Actividades SIPAC.* La Habana : s.n., 2013.
25. **Lorenz, Mark y Kidd, Jeff.** *Object-Oriented Software MetricsObject-Oriented Software MetricsPrentice-Hal994.*
26. **Lorenzo Castro, Fernando. 1999-2000.** *Modelo de datos. Conceptos y clasificación.* 1999-2000.
27. **Mestras, Juan Pavón. 2012-2013.** *Manual Servidores Web – Apache.* Facultad de Informática, Universidad Complutense, Madrid : s.n., 2012-2013.
28. **Moffitt, Jack. 2010.** *XMPP Programming with Javascript® and jQuery.* Indianapolis : s.n., 2010. 978-0-470-54071-8.
29. **Mozilla. 2010.** Mozilla Firefox. *Mozilla Firefox.* [En línea] 2010. [Citado el: 8 de Febrero de 2015.] http://www.mozillaeurope.org/es/firefox/3.0/releasenotes/..
30. **NetBeans.org. 2013.** NetBeans.org. *NetBeans.org.* [En línea] 2013. [Citado el: 9 de Febrero de 2015.] http://NetBeans.org.
31. **Object Management Group. 2001.** OMG Unified Modeling Language Specification. *OMG Unified Modeling Language Specification.* [En línea] 2001. [Citado el: 6 de Febrero de 2015.] http://www.omg.org.
32. **Pérez, Yanelis Lara. 2011.** *Implementación y pruebas de un sistema de notificaciones de*

RHODA 2.1. 2011.

33. **Pressman, Roger. 2005.** *Ingeniería de Software, un enfoque práctico.* La Habana : Félix Varela, 2005.
34. **—. 2010.** *Software Engineering A Practitioner's Approach Seventh Edition.* 2010. ISBN 978-0-07-337597-7.
35. **Real Academia Española. 2000.** Diccionario de la lengua española. *Diccionario de la lengua española.* [En línea] 2000. [Citado el: 19 de Junio de 2015.] http://lema.rae.es/drae/.
36. **Ruiz, Francisco. 2009.** CTR - Computadores y Tiempo real. *CTR - Computadores y Tiempo real.* [En línea] 2009. [Citado el: 4 de mayo de 2015.] http://www.ctr.unican.es/asignaturas/is1/is1-t01-trans.pdf...
37. **Sommerville, Ian. 2005.** *Ingeniería del Software. Séptima edición.* Madrid, España : E.Pearson Educación,, 2005. ISBN: 84-7829-074-5..
38. **Unidad docente de Ingeniería del software.** *Patrones del "Gang of Four.* Facultad de informática - Universidad Politécnica de Madrid, España : s.n.
39. **Universidad de las Ciencias Informáticas. 2014.** *Metodología de desarrollo para la actividad productiva de la UCI para la Actividad productiva de la UCI .* 2014.
40. **Visconti , Marcello y Astudillo, Hernán. 2004.** *Fundamentos de Ingeniería de Software.* 2004.
41. **Visual Paradigm. 2010.** Visual Paradigm for UML - UML tool for software application development. *Visual Paradigm for UML - UML tool for software application development.* [En línea] 2010. [Citado el: 9 de Febrero de 2015.] http://www.visual-paradigm.com/product/vpuml/..
42. **Wellings, Burns. 1997.** *Real-time systems and programming languages.* Londres : London Addison Wesley, 1997.
43. **xmpp.org. 2010.** The XMPP Standards Foundation. *The XMPP Standards Foundation.* [En línea] 2010. [Citado el: 18 de Junio de 2015.] http://xmpp.org/.
44. **xmpphp. 2010.** The xmpphp library. [En línea] 2010. [Citado el: 9 de Febrero de 2015.] http://code.google.com/p/xmpphp.
45. **Zend Technologies. 2006.** Zend Framework. *Zend Framework.* [En línea] 2006. http://framework.zend.com/manual/1.5/en/introduction.html.
46. **Zimbra Inc. 2014.** *Zimbra Web Client User Guide.* Texas : s.n., 2014.

ANEXOS

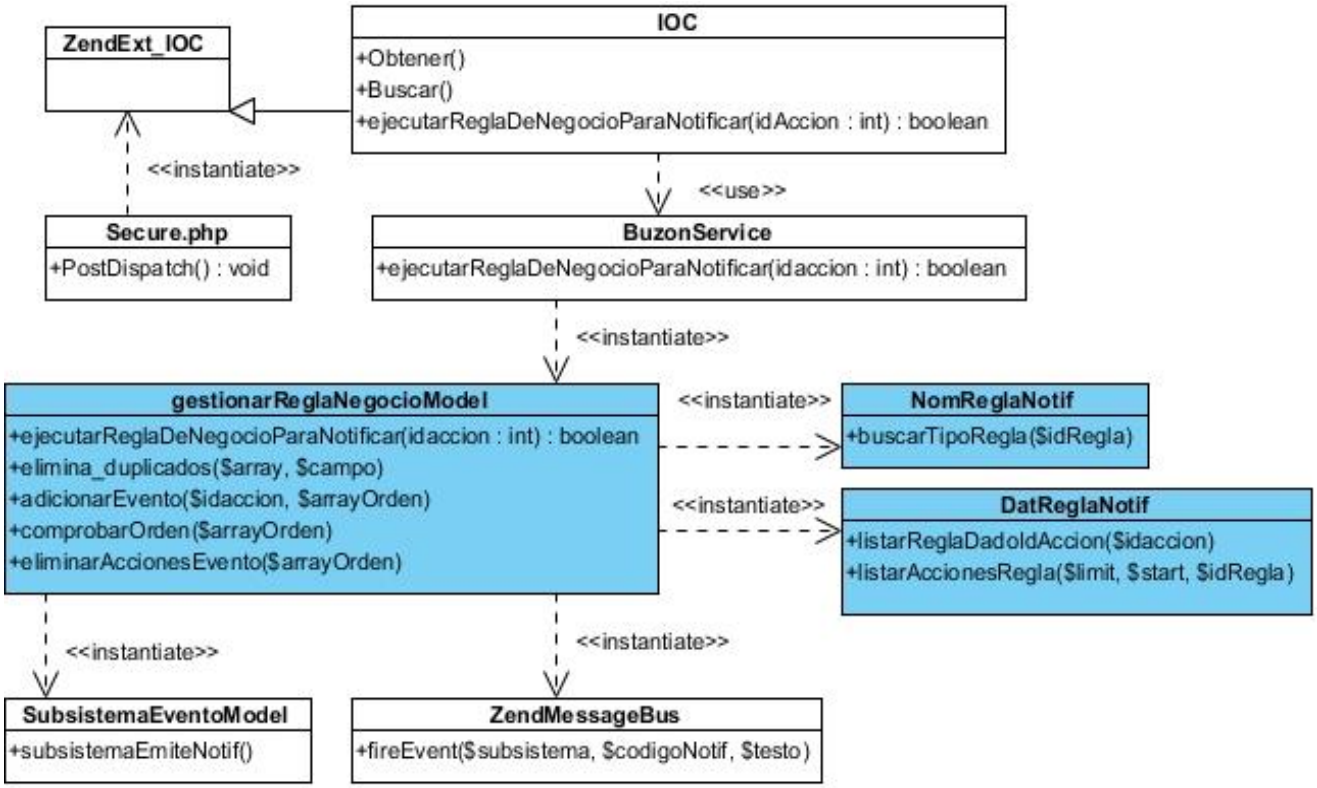


Figura 29: Diagrama de clases de Ejecutar Regla de Negocio ParaNotificar

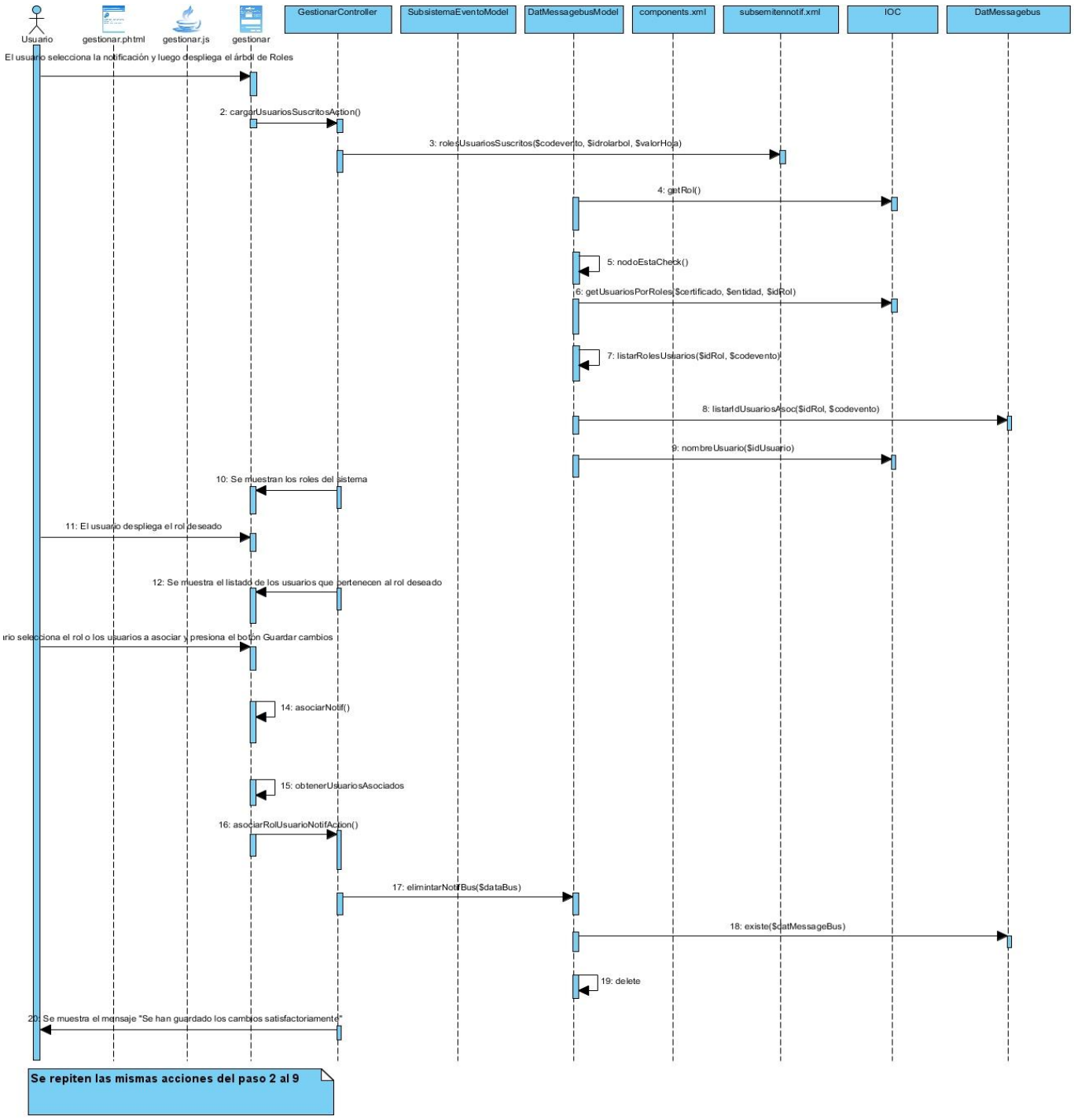
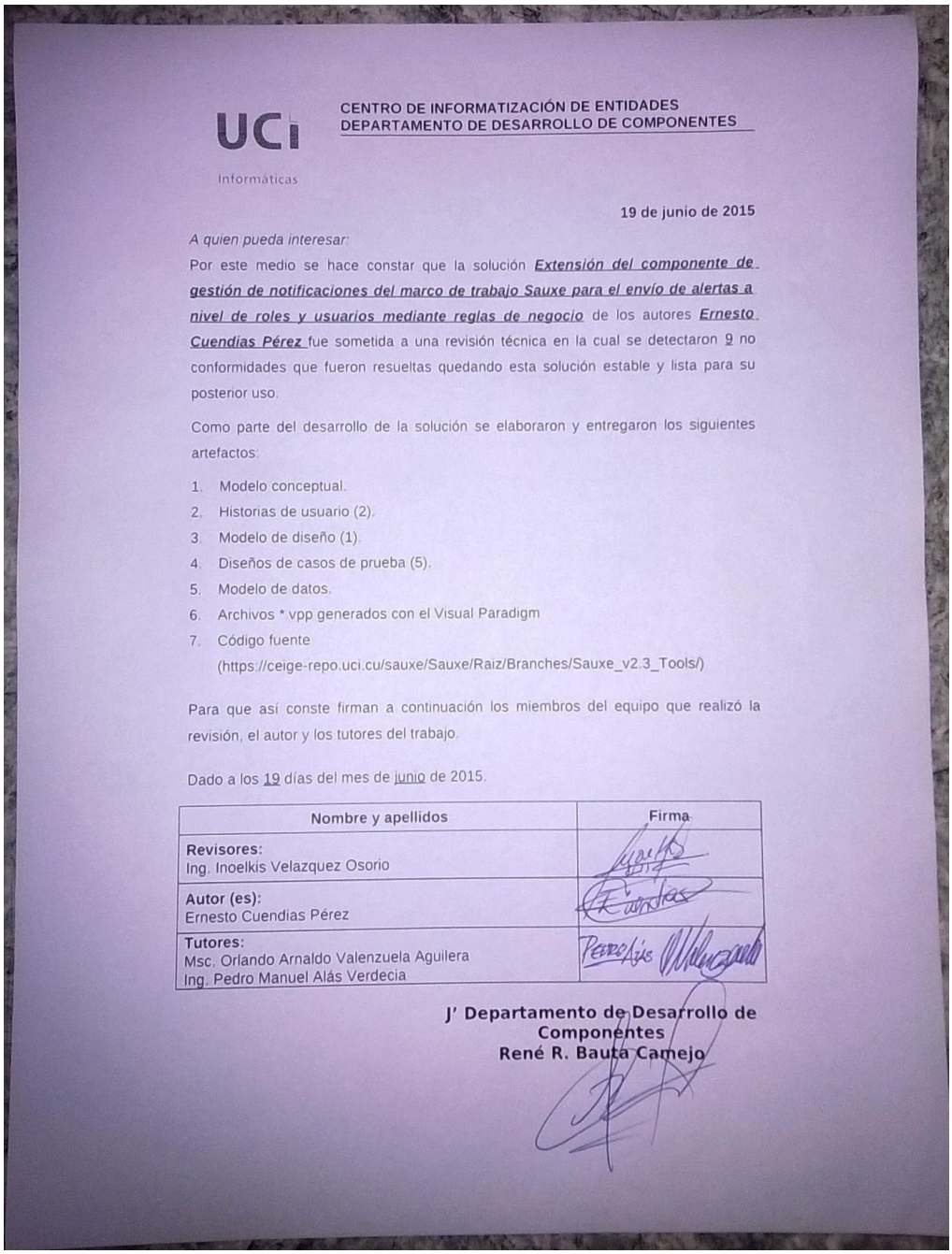


Figura 30 Diagrama de secuencia de Desasociar usuario de notificaciones



UCI
Informáticas

CENTRO DE INFORMATIZACIÓN DE ENTIDADES
DEPARTAMENTO DE DESARROLLO DE COMPONENTES

19 de junio de 2015

A quien pueda interesar:

Por este medio se hace constar que la solución ***Extensión del componente de gestión de notificaciones del marco de trabajo Sauxe para el envío de alertas a nivel de roles y usuarios mediante reglas de negocio*** de los autores ***Ernesto Cuendias Pérez*** fue sometida a una revisión técnica en la cual se detectaron 9 no conformidades que fueron resueltas quedando esta solución estable y lista para su posterior uso.

Como parte del desarrollo de la solución se elaboraron y entregaron los siguientes artefactos:

1. Modelo conceptual.
2. Historias de usuario (2).
3. Modelo de diseño (1).
4. Diseños de casos de prueba (5).
5. Modelo de datos.
6. Archivos * vpp generados con el Visual Paradigm
7. Código fuente (https://ceige-repo.uci.cu/sauxe/Sauxe/Raiz/Branches/Sauxe_v2.3_Tools/)

Para que así conste firman a continuación los miembros del equipo que realizó la revisión, el autor y los tutores del trabajo.

Dado a los 19 días del mes de junio de 2015.

Nombre y apellidos	Firma
Revisores: Ing. Inoelkis Velazquez Osorio	
Autor (es): Ernesto Cuendias Pérez	
Tutores: Msc. Orlando Arnaldo Valenzuela Aguilera Ing. Pedro Manuel Alás Verdecia	

J' Departamento de Desarrollo de Componentes
René R. Bauta Camejo

Figura 31: Acta de aceptación de la aplicación