Universidad de las Ciencias Informáticas

Facultad 2

Trabajo de Diploma para optar por el Título de

Ingeniero en Ciencias Informáticas

Desarrollo del componente de notificaciones 2.0 para el Sistema de Información Hospitalaria del Centro de Informática Médica

Autor: Jorge Joel Yong Ochoa

Tutores: Ing. Nadiezka Milan Cristo

Ing. Juan Manuel García Orduñez

Co-tutor: Ing. Alexei Darias Jojorina

La Habana, Junio 2014

"Año 56 de la Revolución"

Declaración de autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.
Para que así conste firmamos la presente a los 25 días del mes de Junio del año 2014.

Jorge Joel Yong Ochoa

_____

Firma del autor

Ing. Nadiezka Milan Cristo

_____

Firma del tutor

Ing. Juan Manuel García Orduñez

_____

Firma del tutor

Ing. Alexei Darias Jojorina

_____

Firma del tutor

DATOS DE CONTACTO

Ing. Nadiezka Milan Cristo

Profesora con categoría docente Asistente. Graduada en el año 2007 de Ingeniería en Ciencias Informáticas en la Universidad de las Ciencias Informáticas. Jefa del Departamento de Sistemas de Gestión Hospitalaria. Ha impartido las asignaturas: Inteligencia Artificial, Práctica Profesional y Gestión de Software.

Correo electrónico: nmilan@uci.cu

Ing. Juan M. García Orduñez

Profesor con categoría docente Instructor. Graduado de Ingeniería en Ciencias Informáticas en el año 2009 en la Universidad de las Ciencias Informáticas. Desde que se graduó se vinculó a las actividades productivas en el Departamento de Gestión Hospitalaria del Centro de Informática Médica (CESIM), desempeñándose como implementador y jefe del módulo Emergencias del Sistema de Información Hospitalaria del CESIM. Ha impartido las asignaturas: Programación III, Práctica profesional y Gráficos por computadoras.

Correo electrónico: jmgarcia@uci.cu

Ing. Alexei Darias Jojorina

Graduado en el año 2013 de Ingeniería en Ciencias Informáticas en la Universidad de las Ciencias Informáticas. Se encuentra vinculado al Departamento de Gestión Hospitalaria del Centro de Informática Médica (CESIM), desempeñándose como implementador.

Correo electrónico: aleks@uci.cu

AGRADECIMIENTOS

A mi familia por el amor, cariño y apoyo que siempre me han dado.

A los profesores que mostraron interés en mi formación y me ayudaron a ser un mejor estudiante y persona.

A los profesores del proyecto que siempre estuvieron dispuestos a ayudarme en todo momento.

A mis tutores, por todos los dolores de cabeza que me han dado, pero también por todo el tiempo, dedicación y trabajo dedicado a la realización de este trabajo.

A mis amigos, por haber estado siempre en las buenas y las malas y por aguantarme durante todo este tiempo.

A todas las personas que pusieron su granito de arena en este trabajo.

¡MUCHAS GRACIAS!

DEDICATORIA

A mi mamá por ir a trabajar aun estando enferma para que no me faltara nada.

A mi papá que siempre ha sabido darme todos sus conocimientos y por el gran esfuerzo que está realizando para asegurarme un futuro mejor.

A mis hermanos y amigos que me han enseñado todo lo que la escuela no me va a enseñar.

RESUMEN

En el ámbito de la informática se ha hecho frecuente el uso de sistemas de notificaciones, puesto que estos brindan a los usuarios información útil para la ejecución de las actividades y la toma de decisiones. El Centro de Informática Médica (CESIM) desarrolla un Sistema de Información Hospitalaria (HIS por sus siglas en inglés) que permite gestionar los diferentes procesos de una institución hospitalaria. Este software médico cuenta con dos componentes de notificaciones, uno de notificaciones internas y otro de notificaciones externas, los cuales tienen funcionalidades diferentes y sus configuraciones están en distintos lugares, pero sus propósitos son los mismos: alertar a los usuarios ante determinadas situaciones. El presente trabajo tiene como objetivo integrar los componentes de notificación interna y externa del HIS del CESIM, para lograr el bajo acoplamiento entre los módulos del sistema. Para la realización del componente se utilizó JBoss Developer Studio como herramienta de desarrollo, Java como lenguaje de programación, PostgreSQL como sistema gestor de base de datos y Visual Paradigm en el modelado. Además se hizo uso de varios frameworks y librerías, basándose en el patrón Modelo-Vista-Controlador. El desarrollo fue guiado por la metodología Proceso Unificado de Desarrollo (RUP, por sus siglas en inglés), la cual se apoyó en el Lenguaje de Modelado Unificado (UML, por sus siglas en inglés). Luego de concluir el trabajo se obtuvo como resultado un componente único, funcional y configurable, capaz de enviar notificaciones a los roles y usuarios del HIS.

Palabras claves: Centro de Informática Médica, componente, notificaciones, Sistema de Información Hospitalaria, usuarios.

ÍNDICE

INTRODUCCIÓN

La información es uno de los recursos más valiosos que existen, tanto es así que hay toda una era llamada Era de la Información y las Telecomunicaciones, que empieza con la invención del telégrafo eléctrico y el teléfono y que alcanza su clímax con la explosión de internet en la década de los 90. Encontrar o generar la información adecuada a tiempo proporciona una ventaja. Esta es una de las premisas por la que surge la informática, conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático de la información por medio de ordenadores (1). Pero en la actualidad no es posible hablar de información e informática sin asociarlas a las Tecnologías de la Información y las Comunicaciones (TIC).

Las TIC agrupan los elementos y las técnicas usadas en el tratamiento y la transmisión de las informaciones, principalmente de informática, internet y telecomunicaciones (2). Los países y organizaciones del mundo han visto el potencial y el impacto de las TIC en la sociedad y han hecho una fuerte apuesta en su desarrollo y generalización. En Cuba la Revolución ha hecho un gran esfuerzo por promover la informatización de la sociedad, siendo uno de los mayores logros la creación de la Universidad de las Ciencias Informáticas (UCI) en el año 2002.

La UCI cuenta con varios centros de desarrollo de software entre los que se encuentra el Centro de Informática Médica (CESIM). Como parte de las estrategias de negocios de este centro se desarrolla el Sistema de Información Hospitalaria (HIS por sus siglas en inglés) del CESIM. Un HIS puede definirse como un sistema integrado, que procesa, reinterpreta, almacena, comunica y recupera datos médicos y administrativos, respaldando así las exhaustivas necesidades de información presentes en los hospitales. Además proporcionan la gestión de pacientes, clínica, auxiliar y financiera (3). El HIS del CESIM está conformado por 17 módulos los cuáles satisfacen las distintas necesidades presentes en los departamentos hospitalarios. Además cuenta con dos componentes de notificaciones: uno de notificaciones internas y otro de notificaciones externas los cuales mediante el uso de avisos ayudan en la comunicación entre las distintas áreas hospitalarias.

Estos componentes permiten notificar a los pacientes y al personal sanitario registrados en el HIS, permitiendo que estos estén al tanto de los eventos en los que están involucrados. Para cumplir con sus objetivos el componente de notificaciones internas hace uso de avisos que son mostrados dentro del propio HIS, permitiendo que los usuarios puedan ver el objeto de interés en el mismo mensaje. Por otro

lado el componente de notificaciones externas hace uso del correo electrónico, de la telefonía móvil y de la radio-mensajería (beepers) como medios de transmisión.

Sin embargo estos componentes de notificaciones son independientes, sus configuraciones son diferentes y se encuentran separadas dentro del sistema; dificultando las tareas de notificación, configuración y soporte por los desarrolladores del mismo. Además no existe un mecanismo que permita notificar los cambios generados en la información en un período de tiempo determinado. Esto se visualiza cuando un lote de medicamentos está cerca de vencerse donde el administrador del área de almacén no está al tanto de esta situación si antes no chequea periódicamente las fechas de vencimiento de cada lote.

Los componentes de notificaciones no cuentan con mecanismos para avisar de forma automática a los usuarios interesados en la disponibilidad de cierta información generada dentro del sistema. Estos usuarios tienen que esperar a que los involucrados en el proceso notifiquen la información, cuando estos últimos no necesariamente están al tanto de la importancia y urgencia de la misma. Esto ocurre cuando un médico o paciente desea conocer el estado de los resultados de sus exámenes de laboratorio. Otra situación es que el componente de notificaciones externas necesita de la intervención del usuario en la redacción de las notificaciones, posibilitándole cometer errores ortográficos, gramaticales o el envío de una notificación a un destinatario equivocado.

Actualmente la implementación de las notificaciones está embebida dentro de los módulos que las realizan, por lo que existe una dependencia hacia el código de estos cuando se quiere modificar, eliminar o adicionar nuevas notificaciones. Esto atenta contra la modularidad (cada módulo debe ser independiente y capaz de realizar su objetivo) y el bajo acoplamiento (debe existir una mínima dependencia entre los módulos) del sistema ya que los hospitales no siempre cuentan con todas las áreas de atención. Ejemplo de esta situación ocurre entre los módulos Farmacia y Consulta externa donde el primero es el encargado de gestionar el tipo de notificación “Solicitud de medicamentos”. Sin embargo esta responsabilidad debe ser del módulo Consulta externa ya que es el que conoce sobre que medicamentos deben generarse este tipo de notificación.

Teniendo en cuenta la situación antes expuesta se plantea como **problema a resolver:** Las insuficiencias en los componentes de notificación interna y externa del HIS del CESIM afectan el bajo acoplamiento entre los módulos del sistema.

Definiendo como **objeto de estudio** las notificaciones en sistemas informáticos, enfocándose en el **campo de acción** las notificaciones internas y externas del HIS del CESIM.

Para dar solución al problema identificado se define como **objetivo general:** Integrar los componentes de notificación interna y externa del HIS del CESIM para lograr el bajo acoplamiento entre los módulos del sistema.

Con el propósito de dar cumplimiento al objetivo general anteriormente propuesto se identifican las siguientes **tareas de la investigación:**

1. Analizar las tendencias actuales para el envío de notificaciones en los sistemas informáticos.
2. Asimilar las tecnologías, arquitectura y diseño definidos por el departamento de Sistemas de Gestión Hospitalaria para el desarrollo de sus aplicaciones.
3. Obtener los artefactos relacionados con los flujos de trabajo de “Modelado de Negocio”, “Gestión de requerimientos”, “Diseño” e “Implementación”.
4. Implementar las funcionalidades necesarias para la integración de los componentes de notificaciones del Sistema de Información Hospitalaria del CESIM.

Para llevar a cabo la investigación propuesta se han utilizado los siguientes métodos científicos de investigación:

Métodos teóricos: Permiten revelar las relaciones esenciales del objeto de investigación, no observables directamente. Participan en la etapa de asimilación de hechos, fenómenos y procesos y en la construcción del modelo e hipótesis de investigación. Los siguientes métodos teóricos sustentan la investigación. (3)

- **Histórico-lógico:** Su empleo permitió el desarrollo evolutivo y coherente en el estudio de los patrones de diseño, herramientas y sistemas para el desarrollo de los artefactos que proponen los flujos estudiados.
- **Analítico-sintético:** Permitió mediante el análisis descomponer en múltiples partes y relaciones el conocimiento obtenido en la bibliografía consultada. Entonces mediante la síntesis se descubrió las relaciones esenciales y las características generales entre ellas. Se utilizaron en la realización de los artefactos propuestos durante el desarrollo de la presente investigación, determinando los aspectos esenciales y arribando a conclusiones prácticas y teóricas.
- **Modelación:** Mediante la utilización de este método se crearon abstracciones que permitieron explicar la realidad, por ejemplo, todos los modelos y diagramas presentados. En la presente

investigación se utilizó este método científico en la representación de los modelos de dominio, de casos de uso del sistema y de despliegue, entre otros.

Métodos empíricos: Como parte de la investigación es necesario determinar el método de recolección de datos y tipo de instrumento que se utilizará, por lo cual deberán tomarse en cuenta especialmente los objetivos y las categorías del estudio expuestos con anterioridad. (3)

- **Método observación:** Permitió conocer la realidad mediante la percepción directa del objeto de investigación. En la presente investigación fue utilizada la observación científica ya que es consciente y es orientada hacia un objetivo específico poniéndose de manifiesto en la formulación del problema a investigar, en el estudio del estado del arte y muchas etapas de la investigación.
- **Método entrevista:** Para la presente investigación se utilizó el método de la entrevista no estructurada durante el levantamiento de requisitos con el cliente, proporcionando los datos necesarios y de interés para las posteriores etapas de la misma. (ver Anexo 1)

Con el desarrollo del componente de notificaciones se esperan obtener los siguientes **beneficios:**

- Integrar todas las características de los componentes de notificaciones presentes en el sistema.
- Disminuir la dependencia entre los módulos del HIS durante el proceso de envío de notificación.

El documento que se presenta cuenta con la siguiente **estructura:**

Capítulo 1 Fundamentación teórica: Se realiza un estudio de algunos sistemas de avisos presentes en los sistemas de gestión. Se presentan los principales conceptos que constituyen las bases para el desarrollo del componente de notificaciones y se argumentan las tecnologías, metodologías y herramientas de desarrollo utilizadas.

Capítulo 2 Características del sistema: Se describen las principales características del sistema. Se presentan los modelos de los distintos procesos necesarios para el desarrollo del componente, además de la definición de los requisitos funcionales y no funcionales del sistema.

Capítulo 3 Análisis y diseño del sistema: Se describe y fundamenta la arquitectura utilizada. Además se realiza el modelo de diseño de las funcionalidades a desarrollar. Se muestran los diagramas de clases correspondientes para modelar el comportamiento del sistema y se definen las estrategias de integración a tener en cuenta.

Capítulo 4 Implementación: Se introduce el flujo de trabajo de implementación, partiendo de los resultados obtenidos en el diseño. Se detalla el Modelo de Datos, en el que se ve la estructura donde se almacena toda la información requerida en el sistema y se exponen aspectos referentes a la seguridad del sistema, las estrategias de codificación, así como la forma en que se tratarán los errores.

CAPÍTULO 1 FUNDAMENTACIÓN TEÓRICA

En este capítulo se describen los principales conceptos que constituyen las bases para el desarrollo del componente de notificaciones. Se realiza un breve análisis de algunos sistemas de notificaciones presentes en sistemas informáticos, obteniendo algunas de sus mejores características así como las comunes en todos ellos. Además se argumentan las tecnologías, metodologías y herramientas de desarrollo utilizadas.

1.1 Sistemas de notificaciones existentes

En la actualidad las notificaciones tienen un gran auge en los sistemas informáticos. Debido a su extendido uso se hace un estudio, obteniendo características y funcionalidades comunes que estos presentan en el envío de notificaciones. A continuación se describen algunos de los sistemas encontrados.

Sistema de Notificación de Sucesos (SNS): Sistema de notificación de la Agencia Estatal de Seguridad Aérea (AESA). Hace uso de un apartado de correo, fax y del correo electrónico como sus principales canales de notificación. Además está orientado a la prevención, donde la notificación de los sucesos es una obligación de los agentes involucrados en el transporte aéreo (usuarios) existiendo garantías de confidencialidad y protección hacia el notificante. SNS tiene como algunas de sus funciones principales la recepción, procesamiento y análisis de todas las notificaciones que se reporten y la elaboración de informes estadísticos y técnicos. Además propone medidas preventivas y correctoras y la explotación y difusión de información estadística a todos aquellos interesados que así lo soliciten. (6)

Aunque la obligación de notificar no se extiende a todos los profesionales, en este sistema se admiten notificaciones voluntarias procedentes de cualquier persona u organización que realice actividades relacionadas con la aviación. AESA anima a todo el personal que lo desee a reportar directamente al SNS, en beneficio de la seguridad operacional. El SNS no admite notificaciones anónimas, ya que dificultan las labores de verificación de los sucesos. Los formularios recibidos se procesan extrayendo la información que pueda identificar al notificante y, posteriormente, se destruyen. De este modo se garantiza la confidencialidad y se protegen las identidades de los notificantes.

Encounter Notification System (ENS): Sistema de notificación lanzado por Chesapeake Regional Information System for Our Patients (CRISP) en agosto del 2013 en el estado de Maryland, en los Estados

Unidos. Hace uso de notificaciones electrónicas en tiempo real cuando los pacientes son ingresados, dados de alta o transferidos dentro de un hospital. Utiliza mensajes ADT (Admission, Discharge and Transfer) que son un grupo de mensajes definidos en el estándar HL7 (Health Level Seven) para notificar a los proveedores de atención primaria cuando los pacientes son hospitalizados o visitan departamentos de emergencia. Los mensajes son enviados en formato PDF a bandejas de entradas electrónicas o son guardados directamente en su historia clínica electrónica como mensajes HL7. Este sistema posibilita a los médicos seleccionar a los pacientes sobre los cuales recibir notificaciones. Además hace uso del protocolo Direct Project para transportar de forma segura las notificaciones. ENS solo se ofrece en colaboración con los hospitales participantes, sin costo para los proveedores ambulatorios. (7)

Orion Health Notifications: Sistema de notificaciones privativo desarrollado por Orion Health. Se basa en el uso del correo electrónico, Short Message Service (SMS), Instant Message (IM) o a través de mensajes de voz. Este permite distribuir alertas relevantes e información clínica mientras brinda a los proveedores un control total de las notificaciones a recibir. Garantiza que cada proveedor tenga acceso a un amplio conjunto de datos en tiempo real en el tratamiento de sus pacientes. Además brinda características que permiten al usuario subscribirse a eventos específicos y ser notificado cuando ocurre alguno de estos. Tales eventos pueden incluir la disponibilidad de los resultados de laboratorio o la admisión de un paciente. Este sistema está implantado en los Orion Health HIS desplegados en más de 30 países del mundo. (8)

Notifi Critical Event Notifications: Solución informática privativa desarrollada en el año 2012 por HIT Application Solutions en el estado de Pensilvania, en los Estados Unidos. Como vías de notificación hace uso del fax, correo electrónico, SMS y mensajes de voz. Este sistema alerta a los socios comunitarios en el proceso continuo de atención a pacientes envueltos en diferentes eventos. Los más significativos son la admisión y el alta en el servicio de urgencias, o en el servicio médico del hospital. Dependiendo del tipo de notificación, Notifi puede proporcionar automáticamente informaciones tales como el motivo de la consulta, el diagnóstico de admisión, resultados significativos de laboratorios o radiológicos, el nombre y datos de contacto del médico del hospital y la información del paciente, entre otros. Esta información estará disponible para los médicos de atención primaria, especialistas y casas de salud en tiempo real, mientras es capturada por el HIS. (9)

Componente de notificaciones internas del HIS de CESIM: Componente de software integrado al HIS del CESIM en el año 2008, cuyas funcionalidades son accesibles desde cualquier módulo del sistema. Permite el envío de notificaciones automáticas a roles y usuarios del sistema mediante avisos que son mostrados al usuario dentro del propio HIS. Estas notificaciones internas aparecen en la esquina inferior derecha de la pantalla en un pequeño mensaje que desaparece a los tres segundos. Además permite darle un seguimiento a una información más detallada, permitiendo ver los datos del objeto sobre el cual se generó la notificación.

Componente de notificaciones externas del HIS de CESIM: Componente de software integrado al HIS del CESIM en el año 2013, cuyas funcionalidades son accesibles desde cualquier módulo del sistema. Permite enviar notificaciones a roles y usuarios del sistema, las cuales llegan por las vías del correo electrónico, la mensajería móvil o el beeper. Además da la posibilidad de remitir mensajes SMS a los pacientes registrados en la aplicación. Con el objetivo de notificar cualquier evento o situación que se requiera, se brinda la opción de introducir direcciones o números específicos (no gestionados por el HIS) como destinatarios. Para el envío de notificaciones el administrador del sistema debe configurar previamente los parámetros de conexión en cada entidad del HIS. En aras de dar un posterior seguimiento a la información transmitida, se registran trazas por cada envío realizado. El administrador del sistema puede generar reportes con los datos de dichas trazas.

Al realizar el estudio se han encontrado sistemas de notificaciones ya implementados con funcionalidades comunes y que marcan las tendencias actuales en el envío de notificaciones. Pero estos son mayormente privativos, no brindan información sobre su diseño e implementación o sobre las herramientas utilizadas en su desarrollo. Sin embargo cuentan con características distintivas, que pueden ser aprovechadas en conjunto para hacer un sistema de notificaciones muy potente. Por estas razones se ha decidido desarrollar un componente de notificaciones con algunas de estas funcionalidades, por ejemplo: un buzón de notificaciones para cada usuario, donde pueda administrar de manera sencilla las notificaciones existentes. La solución a desarrollar integrará los dos componentes existentes en el HIS añadiéndole nuevas características para el envío automático de notificaciones.

1.2 Tendencias y tecnologías actuales a considerar

En este acápite se determina el entorno utilizado en el desarrollo del componente de notificaciones teniendo en cuenta sus características y utilidades. Las tecnologías, herramientas y arquitecturas

empleadas son las definidas por el departamento de Sistemas de Gestión Hospitalaria del CESIM para el desarrollo de los componentes o módulos del HIS, caracterizándose a continuación.

1.2.1 Arquitectura de software

La **arquitectura cliente-servidor** es un modelo para el desarrollo de sistemas de información, en el que las transacciones se dividen en procesos independientes que cooperan entre sí para intercambiar información, servicios o recursos. Se denomina cliente al proceso que inicia el diálogo o solicita los recursos y servidor, al proceso que responde a las solicitudes. Es uno de los modelos de interacción más utilizados entre aplicaciones en una red. (10)

La arquitectura cliente-servidor a su vez utiliza un sistema de división de las aplicaciones, destacándose la **arquitectura en tres capas**, donde una capa servirá para guardar los datos (base de datos), otra para centralizar la lógica de negocio (modelo) y por último una interfaz gráfica que facilite al usuario el uso del sistema. Esta arquitectura nos brinda algunas ventajas como son la centralización de los aspectos de seguridad y la transaccionalidad y la disminución de los costes de mantenimiento, entre otras. (11)

El patrón de arquitectura **Modelo-Vista-Controlador (MVC)** se basa en separar el modelo de datos de la aplicación de su representación de cara al usuario y de la interacción de éste con el sistema. Divide la aplicación en tres partes fundamentales: el modelo, que contiene la lógica de negocio de la aplicación, la vista, que muestra al usuario la información que éste necesita y el controlador, que recibe e interpreta la interacción del usuario. Este patrón ha demostrado ser muy apropiado para las aplicaciones web y se adapta a las tecnologías proporcionadas por la plataforma J2EE (Java 2 Platform Enterprise Edition), brindando importantes características como son la reutilización de código y la separación de conceptos, las cuáles buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento. (11)

1.2.2 Lenguaje de programación Java

Java: Lenguaje orientado a objetos, distribuido, interpretado, sólido, seguro, de arquitectura neutral, de alto desempeño, de multihilos y dinámico (12). Java es la base de casi todos los tipos de aplicaciones en red y es ampliamente usado en el desarrollo y suministro de aplicaciones móviles, juegos, contenido basado en web y software de empresa. Permite desarrollar y desplegar de un modo eficiente interesantes aplicaciones y servicios. Con un conjunto integral de herramientas, un ecosistema maduro y un sólido

rendimiento, Java ofrece portabilidad de aplicaciones incluso entre los entornos informáticos más dispares. (13)

1.2.3 Capa de presentación

JavaServer Faces Technologies (JSF): Establece el estándar en el desarrollo de las interfaces de usuario del lado del servidor. Incluye un conjunto de API (del inglés Application Programming Interface) para representar los componentes de la interfaz de usuario y para gestionar sus estados, manejando eventos y validando entradas, definiendo la navegación por páginas y dando soporte a la internacionalización y la accesibilidad. (14)

RichFaces: Es un framework avanzado de interfaz de usuario para integrar fácilmente las capacidades de Ajax en aplicaciones que usan JSF. Sus principales ventajas son la facilidad de uso, la optimización del rendimiento, los recursos dinámicos y el desarrollo de componentes enriquecidos para JSF. (15)

Ajax4Jsf: Es un framework, el cual, cuando es incluido en una aplicación web adiciona soporte Ajax en los componentes JSF. Por lo tanto, todos los componentes en la librería Ajax4Jsf no son más que la extensión de componentes JSF. El uso transparente de código JavaScript junto con XML Http Request Objects es atendido por el propio framework, por lo que el desarrollador no tiene que mezclar código JavaScript en su aplicación. (16)

Facelets: Es un poderoso pero ligero lenguaje de plantillas empleado para desarrollar las vistas JSF usando estilos de plantillas HTML (Hypertext Markup Language) y para construir los árboles de componentes. Facelets es parte de la especificación de JSF y es además una de las tecnologías de presentación preferida en el desarrollo de aplicaciones basadas en la tecnología JSF. (17)

XHTML: Lenguaje de Marcado de Hipertexto Extensible. Es una versión más estricta y limpia de HTML, que nace precisamente con el objetivo de remplazar a HTML ante su limitación de uso con las cada vez más abundantes herramientas basadas en XML. XHTML está orientado al uso de un etiquetado correcto, requiriendo una estructuración coherente dentro del documento donde se incluyen elementos correctamente anidados, etiquetas en minúsculas, elementos cerrados correctamente, atributos de valores entre comillas, entre otros. Además permite contener otros lenguajes como MathML (Mathematical Markup Language) y SMIL (Synchronized Multimedia Integration Language). (18)

1.2.4 Capa de negocio

Seam: Es un framework de aplicación para Java EE. Este define un modelo de componente uniforme para toda la lógica de negocio de la aplicación. Integra dos de las principales características presentes en Java EE 5 que son JSF y EJB 3.0 (Enterprise Java Beans 3.0). Además soporta las mejores soluciones JSF para la interfaz de usuario como son RichFaces e ICE faces, agregando soporte Ajax sin la necesidad de escribir código JavaScript. Proporciona opcionalmente, una gestión transparente de los procesos de negocio vía JBPM (Java Business Process Management) implementando fácilmente complejos flujos de trabajo, así como la integración de JPA (Java Persistence API) y de Hibernate3 para la persistencia de los datos. (19)

1.2.5 Capa de acceso a datos

Hibernate: Herramienta de mapeo objeto-relacional (ORM, por sus siglas en inglés) de código abierto para ambientes Java. Hibernate se hace cargo de mapear clases Java hacia tablas de bases de datos y viceversa. Además provee facilidades de recuperación y consultas de datos mediante el lenguaje HQL (Hibernate Query Language), el cual es completamente orientado a objetos. (20)

EJB3: Enterprise Java Beans 3.0 es una revisión profunda y una simplificación de la especificación de EJB. Su objetivo es disminuir las tareas del programador, facilitar el desarrollo guiado por pruebas y enfocarse más en escribir clases simples sin dependencia de frameworks, conocidas como POJO (Plain Old Java Objects), en vez de complejas APIs de Enterprise Java Beans. Además hace uso de Hibernate como motor de Java Persistence para mantener sincronizadas las clases POJO con la base de datos automáticamente. (21)

1.3 Tecnologías horizontales

Java EE 5: Es la versión empresarial de Java después de J2EE 1.4. Es una plataforma de programación que simplifica el desarrollo de aplicaciones empresariales basadas en la Web. Disminuye la necesidad de programar mediante la creación estandarizada de componentes modulares reutilizables. Además maneja muchos aspectos de la programación automáticamente. (22)

JBoss Application Server: Es un servidor de aplicaciones con certificación Java EE 5, por lo que soporta todas las especificaciones correspondientes, incluyendo servicios adicionales como clusterizar, carga en memoria caché y persistencia. También soporta EJB3. Cuenta con un conjunto de componentes claves como son: JBoss AS 4.2, Hibernate 3.2.4, Seam 2.0. (23)

1.4 Metodologías de desarrollo de software

Las metodologías de desarrollo de software definen quién debe hacer qué, cuándo y cómo. Son un proceso donde se realizan tareas para obtener un resultado a través de un conjunto de actividades definidas previamente. En la actualidad no existe una metodología universal que se le pueda aplicar a todos los proyectos por lo que su selección depende de las características de cada uno de ellos.

Proceso Unificado de Desarrollo (RUP): Proceso de software genérico que provee un enfoque disciplinado en la asignación de tareas y responsabilidades dentro de una organización de desarrollo. Su meta es asegurar la producción de software de muy alta calidad que satisfaga las necesidades de los usuarios finales, dentro de un calendario y presupuesto predecible. Tiene dos dimensiones: una que representa el aspecto dinámico del proceso conforme se va desarrollando (fases, iteraciones, hitos) y otra que representa el aspecto estático del proceso (componentes del proceso, disciplinas, actividades, flujos de trabajo). Se caracteriza por ser guiado por casos de uso, centrado en la arquitectura e iterativo e incremental. (24)

Lenguaje Unificado de Modelado (UML): Se basa en que el modelado es el diseño de software antes de su programación. Permite modelar no solo estructuras de aplicaciones, comportamiento y arquitectura, sino también procesos de negocio y estructuras de datos. Es una de las piezas claves en la arquitectura basada en modelos. Ayuda a especificar, visualizar y documentar modelos de sistemas de software, incluyendo su estructura y diseño. (25)

1.5 Herramientas

JBoss Developer Studio: Es un entorno de desarrollo integrado (IDE) basado en Eclipse. Proporciona soporte para todo el ciclo de vida del desarrollo de software. Este incluye un amplio conjunto de herramientas y da soporte a múltiples modelos de programación y frameworks, incluyendo Java EE 6, RichFaces, JSF, EJB, JPA, Hibernate, Contextos de Inyección de Dependencia (CDI, por sus siglas en inglés), HTML5, y muchas otras tecnologías populares. Está probado y certificado para asegurar que todos sus plug-ins, componentes de tiempo de ejecución y sus dependencias son compatibles entre sí (26). Se utiliza JBoss Developer Studio en su versión 7.1.0 GA dadas las características y ventajas que brinda.

PostgreSQL: Es un sistema de gestión de bases de datos objeto-relacional, con licencia BSD (Berkeley Software Distribution). Cuenta con una arquitectura probada que se ha ganado una sólida reputación de fiabilidad, integridad de datos y exactitud. Funciona en todos los principales sistemas operativos, incluyendo Linux, UNIX y Windows. Es totalmente compatible con ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad). Tiene soporte completo para claves foráneas, uniones, vistas, disparadores y procedimientos almacenados (en varios idiomas). Incluye la mayoría de tipos de datos SQL (Lenguaje de Consulta Estructurado, en español), incluyendo los numéricos, booleanos, cadenas y fechas, entre muchos más. También es compatible con el almacenamiento de grandes objetos binarios, como imágenes, sonidos o vídeos. Cuenta con interfaces de programación nativas para C / C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, entre otros, además de una vasta documentación (27). Se utiliza PostgreSQL en su versión 9.2.4 dadas las características y ventajas que brinda.

Visual Paradigm for UML (VP-UML): Es una herramienta de diseño que soporta el ciclo de vida completo de desarrollo de software. Soporta los estándares de modelado como UML, SysML (Systems Modeling Language), ERD (Entity-Relationship Diagrams), DFD (Diagrama de Flujo de Datos), BPMN (en español, Notación para el Modelado de Procesos de Negocio), entre otros. VP-UML facilita la construcción de software y sistemas que se destacan en la experiencia del usuario mediante el efectivo soporte en la identificación de casos de usos, la recopilación de requisitos, el flujo de eventos y la generación de la especificación de los requisitos, entre otros (28). Se utiliza Visual Paradigm for UML en su versión 8.0 dadas las características y ventajas que brinda.

Los elementos relacionados a la fundamentación teórica sirvieron de soporte a la investigación con el objetivo de solucionar el problema presente. Con el estudio de los sistemas de notificaciones existentes se concluyó que estos cuentan con algunas de las funcionalidades requeridas, pero por ser privativos no permiten ser integrados al sistema. Por tal motivo se decide integrar los dos componentes del HIS del CESIM, manteniendo las características que estos presentan y que son comunes en todos los sistemas estudiados, agregándole nuevas funcionalidades, por ejemplo: permitir la suscripción a eventos sobre los cuales ser notificados. Además con el análisis de las tendencias actuales en el desarrollo de software se fundamentaron la selección de las tecnologías y herramientas que permitirán desarrollar un componente totalmente compatible e integrable al HIS del CESIM.

CAPÍTULO 2 CARACTERÍSTICAS DEL SISTEMA

En este capítulo se abordan los principales aspectos que se tuvieron en cuenta en el desarrollo del componente de notificaciones. Se muestra el modelo de dominio donde se exponen los conceptos relacionados con el sistema, además se detallan los requerimientos funcionales y no funcionales y por último se identifican los casos de uso del sistema obteniendo una descripción detallada del componente.

2.1 Modelo de dominio

El modelo de dominio es una representación visual de los conceptos que se manejan en el dominio del sistema en desarrollo. Estos conceptos no describen componentes de software sino entidades del mundo real que están asociadas al problema en cuestión. Dicho modelo podrá ser utilizado como una base de las abstracciones relevantes en el proceso de construcción del sistema. Aprovechando las bondades de los diagramas UML para representar conceptos, el modelo de dominio se presenta en forma de diagrama de clases donde figuran los principales conceptos y roles del sistema en cuestión. (27)

2.1.1 Conceptos fundamentales del dominio

Usuario: Persona que por medio de un ordenador puede acceder a los recursos y servicios que ofrece el sistema, de acuerdo con los privilegios, permisos y roles asignados.

Administrador: Usuario del sistema encargado de realizar la configuración general del mismo.

Evento: Acción que es generada por el usuario o el propio sistema; el usuario a su vez, puede hacer uso del mismo o ignorarlo.

Notificación: Contenido a transmitir entre un emisor (administrador, médico, almacenero, farmacéutico y demás usuarios del sistema) y un receptor (paciente, médico, farmacéutico, entre otros) en una situación comunicacional. Esta puede ser generada automáticamente a partir de un evento, en tal caso el emisor es el propio sistema.

Mensaje a correo: Contenido a transmitir entre un emisor y un receptor a través del correo electrónico.

Mensaje a beeper: Contenido a transmitir por un emisor, desde el sistema hasta el beeper de un receptor.

Mensaje a teléfono móvil: Contenido a transmitir por un emisor, desde el sistema hasta el teléfono móvil de un receptor.

Adjunto: Archivo o fichero que va o está unido al cuerpo o contenido del mensaje de correo electrónico.

Traza: Registro oficial de eventos durante un rango de tiempo en particular, que pueden ser recolectados y analizados con herramientas y técnicas especiales, lo que deja al descubierto la actividad registrada en el mismo.

2.1.2 Diagrama de modelo de dominio

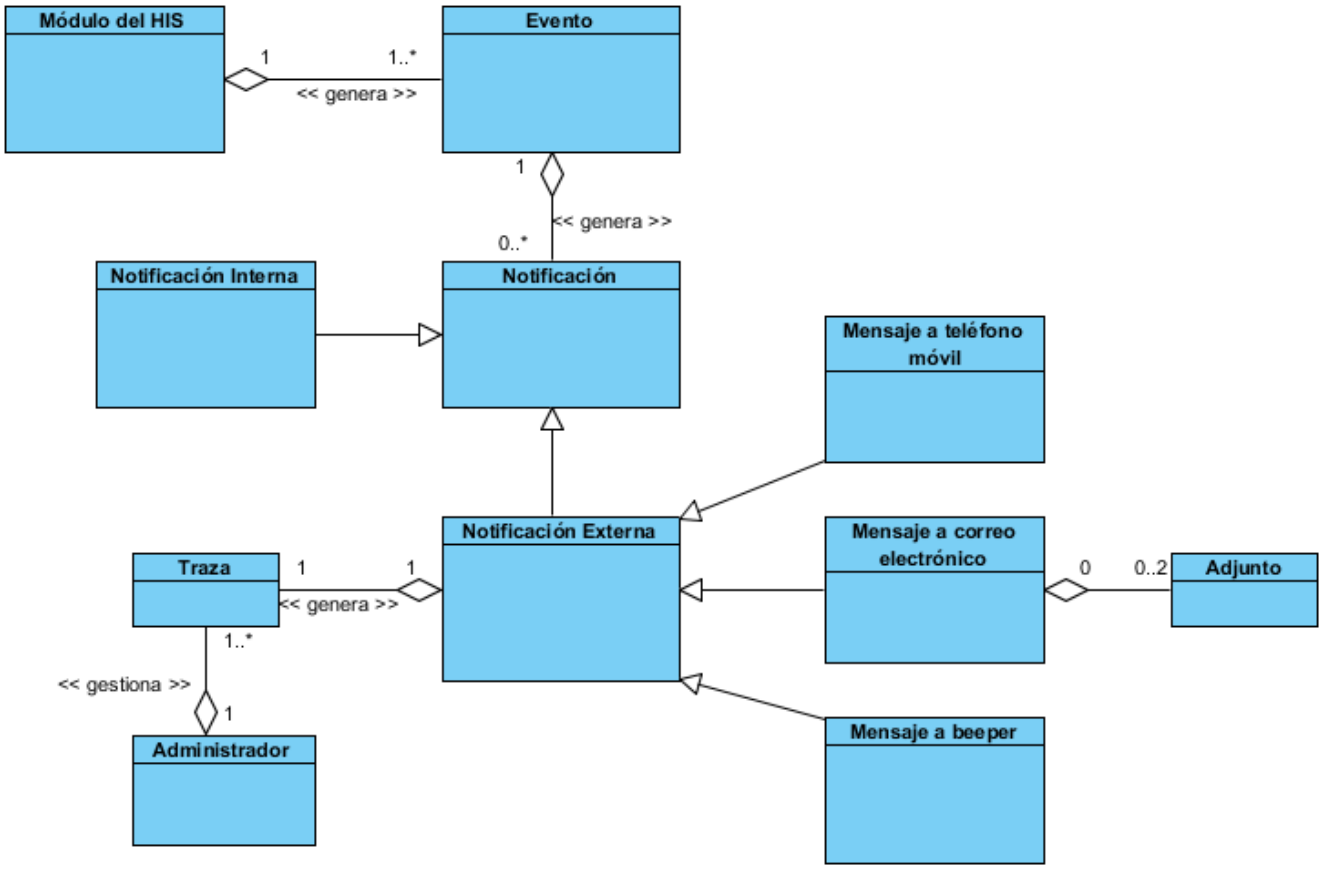

Figura 1. Diagrama de modelo de dominio

En un módulo del HIS se generan varios eventos, donde cada uno de estos puede generar o no notificaciones internas y externas automáticamente. Las notificaciones externas hacen uso de la telefonía móvil, de la mensajería beeper y de los correos electrónicos como medios de transmisión, este último brinda la posibilidad de adjuntar hasta dos archivos en el propio mensaje. Por cuestiones de seguridad el componente de notificaciones externas hace uso de trazas registrando todos los envíos de notificaciones

mediante esta vía. Estas trazas pueden ser gestionadas por un administrador, permitiéndole dar seguimiento a la información enviada.

2.2 Especificación de los requisitos del sistema

2.2.1 Requisitos funcionales del sistema

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir, estos deben ser comprensibles por los clientes, usuarios y desarrolladores, deben tener una sola interpretación y estar definidos en una forma medible y verificable (27). A continuación se muestran los requisitos funcionales definidos:

RF1. Realizar notificación programada

RF2. Realizar notificación mediante suscripción a eventos

RF3. Ver datos de notificación actual

RF4. Buscar notificación

RF5. Ver datos de notificación

RF6. Eliminar notificación

2.2.2 Requisitos no funcionales del sistema

Los requisitos no funcionales representan las condiciones que debe cumplir un sistema para satisfacer las necesidades de un usuario. Estos permiten que la solución se ajuste a las necesidades del usuario y cuente con las características fundamentales de un sistema (tiempo de respuesta, fiabilidad, capacidad de almacenamiento, entre otros) dentro de los límites establecidos. (27) A continuación se muestran los requisitos no funcionales definidos:

1. RNF Usabilidad

El sistema y sus próximos desarrollos estarán guiados por las pautas de diseño de interfaz de usuario, navegación y flujos de trabajo descritos en el documento de arquitectura de información, enriquecido además con la experiencia adquirida en los despliegues de la aplicación. De esta manera se garantiza que los usuarios adquieran las habilidades necesarias para explotarlo en un tiempo reducido:

Usuarios normales: 30 días

Usuarios avanzados: 20 días

2. RNF Fiabilidad

Se mantendrá seguridad y control a nivel de usuario, garantizando el acceso de los mismos sólo a los niveles establecidos de acuerdo a la función que realizan. Se registrarán todas las acciones que se realizan, llevando el control de las actividades de cada usuario en todo momento. Se establecerán mecanismos de control y verificación para los procesos susceptibles de fraude.

El sistema implementará un control de cambios a determinados campos de información (seleccionados por su importancia), de forma tal que sea posible determinar cuáles han sido las actualizaciones (versiones) que se le han realizado. Ninguna información que se haya ingresado en el sistema será eliminada físicamente de la base de datos, independientemente de que para el sistema, este elemento ya no exista.

3. RNF Eficiencia

El sistema respetará buenas prácticas de programación para incrementar el rendimiento en operaciones costosas para la máquina virtual como la creación de objetos. Se deberá usar siempre que sea posible el patrón Singleton, destruir referencias que ya no estén siendo usadas, optimizar el trabajo con cadenas, entre otras buenas prácticas que ayudan a mejorar el rendimiento.

4. RNF Soporte. Restricciones de diseño

El sistema estará dividido en las siguientes capas:

- **Capas físicas**

  Cliente: Computadora con cualquier tecnología o sistema operativo y el navegador Firefox 4.x o superior.

  Servidor de Aplicaciones: Servidor con cualquier tecnología y sistema operativo que soporte la máquina virtual Java Runtime Environment (JRE) 1.6.0_24 o superior y el servidor de aplicaciones JBoss AS 4.2.2.GA.

Servidor de Base de Datos: Servidor con cualquier tecnología y sistema operativo que soporte a PostgreSQL Server 8.4 o superior.

- **Capas lógicas**

Presentación: Contiene todas las vistas y la lógica de la presentación. El flujo web se maneja de forma declarativa y basándose en definiciones de procesos del negocio.

Negocio: Mantiene el estado de las conversaciones y procesos del negocio que concurrentemente pueden estar siendo ejecutados por cada usuario. En los casos de que algún objeto del negocio tenga una interfaz externa, siendo accesible la misma desde sistemas legados o directamente del cliente, se garantiza la seguridad a nivel de objeto y métodos.

Acceso a Datos: Contiene las entidades y los objetos de acceso a datos correspondientes a las mismas. El acceso a datos está basado en el estándar JPA y particularmente en la implementación del motor de persistencia Hibernate.

5. **RNF Soporte. Interfaz**

- **Interfaces de usuario**

Las ventanas del sistema contendrán claro y bien estructurados los datos, además de permitir la interpretación correcta de la información. Todos los textos y mensajes en pantalla aparecerán en el idioma de preferencia del usuario siempre que este sea soportado por la aplicación. El diseño de la interfaz del sistema responderá a la ejecución de acciones de una manera rápida, minimizando los pasos a dar en cada proceso.

- **Interfaces hardware**

Clientes: Instalación de computadoras personales (computadoras personales con al menos 512 Mb de RAM).
Servidores: Instalación de los terminal server (terminal server con al menos 16 Gb de RAM y 1 Tb de HDD).

- **Interfaces de comunicación**

Implementará mecanismos de encriptación de datos para el intercambio de información con sistemas externos. Utilizará un modem o teléfono móvil para la implementación de las notificaciones mediante mensajes SMS.

2.3 Modelo de casos de usos del sistema

El modelo de casos de uso del sistema representa la vista externa del mismo y es utilizado principalmente como contrato entre los desarrolladores y el cliente sobre qué debería y qué no debería hacer el sistema. Este modelo está formado por actores, casos de uso y las relaciones que se establecen entre estos, es decir, representa gráficamente a los procesos y su interacción con los actores.

2.3.1 Definición de los actores del sistema

Actor	Descripción
Usuario	Usuario global que se autentica en la aplicación, el sistema lo valida y le asigna permisos según el rol que tenga definido.
Reloj	Un evento que ocurre cada cierto intervalo de tiempo definido.
Hibernate	La herramienta Hibernate cuando un objeto específico es adicionado, modificado o eliminado.

Tabla 1. Definición de los actores del sistema

2.3.2 Diagrama de casos de uso del sistema

Los diagramas de casos de uso permiten a los clientes y desarrolladores llegar a un acuerdo sobre las condiciones y el alcance del sistema antes de emprender la fase de construcción del mismo.

Diagrama de casos de uso del sistema

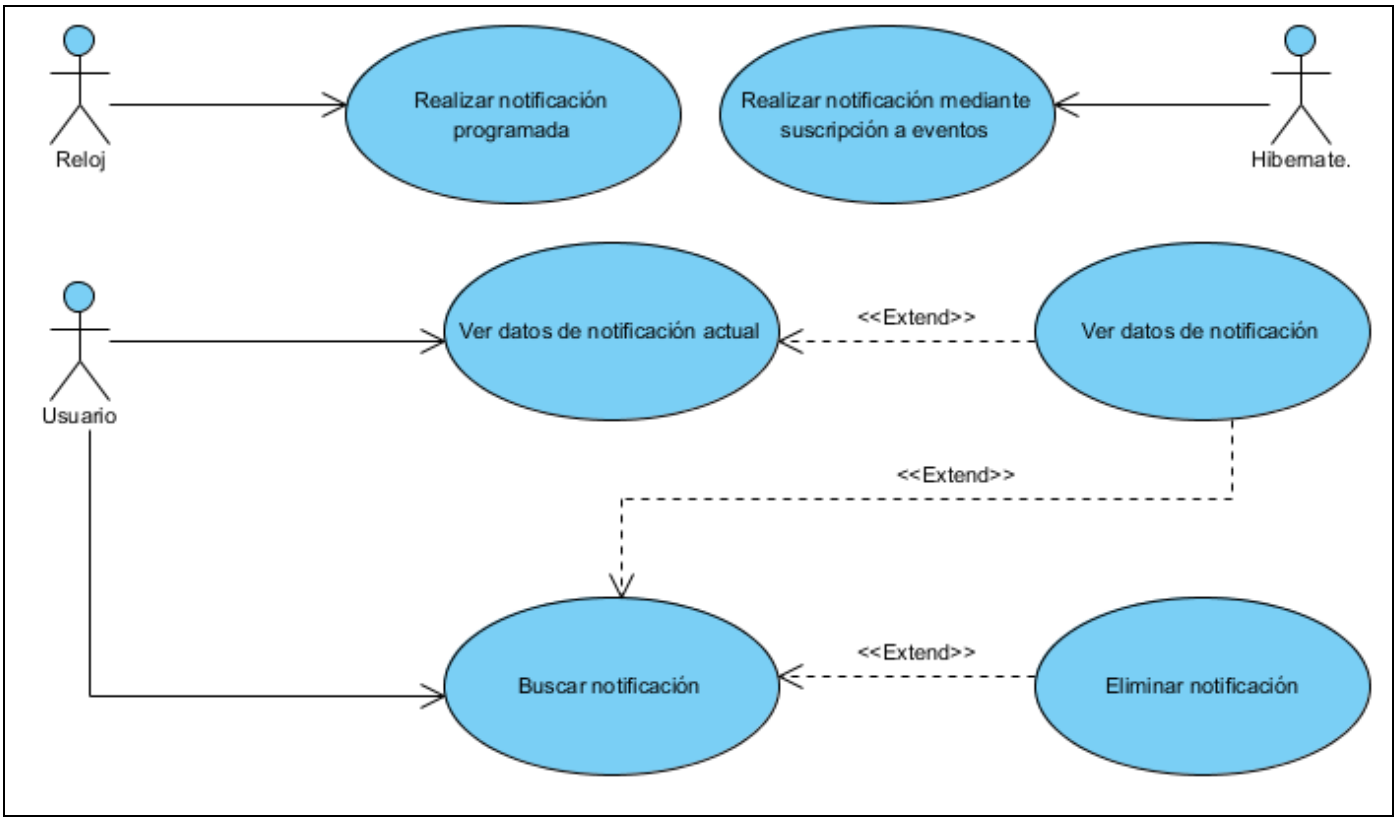

Figura 2. Diagrama de casos de uso del sistema

El componente de notificaciones cuenta con tres actores fundamentales, el reloj del sistema, la herramienta Hibernate y un usuario del sistema. El reloj del sistema se encarga de ejecutar cada cierto intervalo de tiempo definido una tarea programada, la cual se encarga de realizar notificaciones automáticas en caso necesario. La herramienta Hibernate mediante Interceptor interviene cada vez que un objeto de una entidad es creado, modificado o eliminado, entonces se verifica que se cumplan ciertas reglas definidas sobre los datos, en caso positivo se realizan notificaciones automáticas a los roles y usuarios suscritos a dichas reglas.

Un usuario puede buscar notificaciones, filtrándolas por un rango de fecha, y una vez tenga la lista filtrada de notificaciones, puede ver los datos en detalle o borrar la notificación deseada. Cuando el usuario realiza una acción sobre el sistema, en caso de tener alguna notificación nueva, esta le sale en la esquina inferior derecha, permitiendo leer una parte del mensaje y poder ver todos los datos de la notificación en detalle.

2.3.3 Descripción textual de los casos de uso

CU 1. Realizar notificación programada	
Objetivo	Comprobar la existencia de situaciones a notificar
Actores	Reloj
Resumen	El caso de uso se ejecuta en intervalos de tiempo previamente definidos. El sistema comprueba que se cumplan los criterios de ejecución y en caso positivo se verifica si existen datos a notificar, enviando automáticamente una notificación a los roles y usuarios correspondientes. La notificación es emitida hacia el propio sistema, o por las vías de correo electrónico, mensajería de texto y radio-mensajería, según se define para cada tipo de notificación.
Complejidad	Alta
Prioridad	Crítico
Precondiciones	No existen
Poscondiciones	Se enviaron notificaciones automáticamente

Tabla 2. Descripción textual del CU: Realizar notificación programada

CU 2. Realizar notificación mediante suscripción a eventos	
Objetivo	Comprobar la existencia de situaciones a notificar
Actores	Hibernate
Resumen	El caso de uso se ejecuta cuando se insertan, modifican o eliminan datos del sistema, se verifica un conjunto de reglas predefinidas sobre los datos. En caso de cumplirse las reglas se envía automáticamente una notificación a los roles y usuarios correspondientes. La notificación es emitida hacia el propio sistema, o por las vías de correo electrónico, mensajería de texto y radio-mensajería, según se define para cada tipo de notificación.
Complejidad	Alta
Prioridad	Crítico
Precondiciones	No existen
Poscondiciones	Se enviaron notificaciones automáticamente

Tabla 3. Descripción textual del CU: Realizar notificación mediante suscripción a eventos

CU 3. Ver datos de notificación actual	
Objetivo	Comprobar la existencia de notificaciones nuevas
Actores	Usuario
Resumen	El caso de uso se ejecuta cuando se realiza cualquier acción sobre el sistema. Permite ver los datos de la notificación actual para el usuario en un mensaje mostrado en la parte inferior derecha de la pantalla, el caso de uso termina.

Complejidad	Baja		
Prioridad	Crítico		
Precondiciones	El usuario debe estar activo para poder ser notificado mediante este caso de uso		
Poscondiciones	Se vio los datos de una notificación		
Flujo de eventos			
Flujo básico Ver datos de notificación actual			
	Actor	**Sistema**	
1.	El caso de uso inicia cuando se ejecuta cualquier acción sobre el sistema		
2.		Comprueba la existencia de notificaciones nuevas para el usuario y en caso positivo las muestra en un mensaje en la esquina inferior derecha de la pantalla durante 3 segundos. Permite : • Ver datos de notificación. Ver **Alternativa 1: "Ver datos de notificación"**	
3.		Termina el caso de uso	
Flujos alternos			
Alternativa 1. "Ver datos de notificación"			
	Actor	**Sistema**	
1.		Permite ver los datos detallado de la notificación. Ver **CU: "Ver datos de notificación"**	
Relaciones	**CU incluidos**	No existen	
	CU extendidos	Ver datos de notificación en el CU Ver datos de notificación actual	
Requisitos no funcionales	Sin definir		
Asuntos pendientes	Sin definir		

Tabla 4. Descripción textual del CU: Ver datos de notificación actual

CU 4. Buscar notificación	
Objetivo	Buscar notificaciones en un rango de fecha determinado
Actores	Usuario
Resumen	El caso de uso inicia cuando el actor accede a la opción Buscar notificación, el sistema brinda la posibilidad de seleccionar criterios de búsqueda para localizar la notificación, el actor selecciona los datos que considera como criterios para realizar una búsqueda, el sistema busca y muestra las notificaciones que cumplen con los criterios de búsqueda, el caso de uso termina.

Complejidad	Media
Prioridad	Opcional
Precondiciones	No existen
Poscondiciones	Se mostraron las notificaciones que cumplen los criterios de búsqueda.
Flujo de eventos	
Flujo básico Buscar notificación	

	Actor	Sistema
1.	El caso de uso inicia cuando el actor accede a la opción Buscar notificación	
2.		Brinda la posibilidad de seleccionar los criterios de búsqueda: • Fecha inicial • Fecha final Permite: • Buscar notificaciones dado criterios. • Cancelar operación. Ver **Alternativa 1**: **"Cancelar operación"**
3.	Selecciona los datos que considera como criterios para realizar una búsqueda y selecciona la opción de Buscar notificación dado criterios	
5.		Busca los datos de notificación que cumplen con los criterios de búsqueda
6.		Si no se encuentra ninguna notificación que cumpla con los criterios de búsqueda. Ver **Alternativa 2: "No se encontró información que cumpla con los criterios de búsqueda"**
7.		Muestra un listado de notificaciones que cumplen con los criterios de búsqueda, mostrando los siguientes atributos: • Fecha • Mensaje Ordenados descendentemente por la Fecha mostrando la cantidad de elementos configurados para mostrar por página, permitiendo navegar por el resultado
8.		Permite: • Ordenar el resultado por el atributo Fecha,

		de manera ascendente o descendente. Ver **Alternativa 3: "Ordenar el resultado ascendente o descendentemente por un atributo"** • Eliminar notificación. Ver **Alternativa 4: "Eliminar notificación"** • Ver datos de notificación. Ver **Alternativa 5: "Ver datos de notificación"**
9.		Termina el caso de uso
Flujos alternos		
Alternativa 1. "Cancelar operación"		
	Actor	**Sistema**
1.	Selecciona la opción de Cancelar operación	
2.		Regresa a la vista anterior
3.		El caso de uso termina
Alternativa 2. "No se encontró información que cumpla con los criterios de búsqueda"		
	Actor	**Sistema**
1.		Muestra el mensaje de información "No se encontró información que cumpla con los criterios de búsqueda."
2.		Regresa al paso 2 del **Flujo Normal de Eventos**
Alternativa 3: "Ordenar el resultado ascendente o descendentemente por un atributo"		
	Actor	**Sistema**
1.	Selecciona un atributo del resultado para ordenarlo ascendente o descendentemente por el atributo seleccionado	
2.		Reordena y muestra el resultado ascendente o descendentemente por el atributo seleccionado
3.		Regresa al paso 7 del **Flujo Normal de Eventos**
Alternativa 4. "Eliminar notificación"		
	Actor	**Sistema**
1.		Permite eliminar la notificación. Ver **CU: "Eliminar notificación"**
Alternativa 5. "Ver datos de notificación"		
	Actor	**Sistema**
		Permite ver los datos detallados de la notificación. Ver **CU: "Ver datos de notificación"**
Relaciones	**CU incluidos**	No existen

	CU extendidos	Eliminar notificación en el CU Buscar notificación Ver datos de notificación en el CU Buscar notificación
Requisitos no funcionales	Sin definir	
Asuntos pendientes	Sin definir	

Tabla 5. Descripción textual del CU: Buscar notificación

CU 5. Eliminar notificación		
Objetivo	Eliminar una notificación determinada	
Actores	Usuario	
Resumen	El caso de uso inicia cuando el actor selecciona una notificación y accede a la opción Eliminar notificación, el sistema elimina la notificación, el caso de uso termina.	
Complejidad	Baja	
Prioridad	Opcional	
Precondiciones	Para eliminar una notificación, esta debe haber sido seleccionada.	
Poscondiciones	Se eliminó la notificación.	
Flujo de eventos		
Flujo básico Eliminar notificación.		
	Actor	**Sistema**
1.	El caso de uso inicia cuando el actor selecciona una notificación y accede a la opción Eliminar notificación	
2.		Muestra el mensaje de advertencia “Se eliminará la notificación seleccionada. Si selecciona Sí se perderán todos los datos. ¿Desea continuar?“ y permite: • Aceptar la eliminación de la notificación. • Cancelar la operación. Ver **Alternativa 1: “Cancelar operación”**
3.	Selecciona la opción de aceptar la eliminación de la entidad	
4.		Oculta la notificación
5.		Termina el caso de uso
Flujos alternos		
Alternativa 1. “Cancelar operación”		

	Actor	Sistema
1.	Selecciona la opción de Cancelar operación	
2.		Regresa a la vista anterior
3.		El caso de uso termina
Relaciones	**CU incluidos**	No existen
	CU extendidos	No existen
Requisitos no funcionales	Sin definir	
Asuntos pendientes	Sin definir	

Tabla 6. Descripción textual del CU: Eliminar notificación

CU 6. Ver datos de notificación		
Objetivo	Ver los datos detallados de una notificación determinada.	
Actores	Usuario	
Resumen	El caso de uso inicia cuando el actor selecciona una notificación y accede a la opción de Ver datos de notificación, el sistema muestra los datos de la notificación, el caso de uso termina.	
Complejidad	Baja	
Prioridad	Opcional	
Precondiciones	Para ver los datos de una notificación, esta debe estar seleccionada.	
Poscondiciones	Se mostraron los datos detallados de la notificación.	
Flujo de eventos		
Flujo básico Ver datos de notificación.		
	Actor	**Sistema**
1.	El caso de uso inicia cuando el actor selecciona una notificación y accede a la opción de Ver datos de notificación	
2.		Muestra los datos de notificación y permite: • Salir de la vista actual
3.	Selecciona la opción de salir de la vista actual	
4.		Muestra la vista anterior
5.		Termina el caso de uso
Relaciones	**CU incluidos**	No existen
	CU extendidos	No existen
Requisitos no	Sin definir	

funcionales	
Asuntos pendientes	Sin definir

Tabla 7. Descripción textual del CU: Ver datos de notificación

En este capítulo se obtuvo el modelo de dominio, donde se abarcaron las definiciones asociadas al sistema y las relaciones definidas entre ellas, lo que posibilitó un mejor entendimiento del problema y una rápida identificación de las funcionalidades a desarrollar. Fueron enunciados los requisitos funcionales y no funcionales y se definió el modelo de casos de uso del sistema, con el fin de lograr el desarrollo de una aplicación que responda satisfactoriamente a la situación problemática planteada. El diagrama de casos de uso y la descripción de estos permitió obtener la información para un mejor entendimiento de la solución deseada y tener un punto de partida para el desarrollo del componente.

CAPÍTULO 3 ANÁLISIS Y DISEÑO DEL SISTEMA

En este capítulo se describe la concepción arquitectónica definida para el desarrollo de funcionalidades necesarias del componente de notificaciones. Además define los artefactos concernientes al modelo de diseño: diagrama de clases del diseño y de interacción, que son las entradas del modelo de implementación y se describen las clases asociadas a cada diagrama representado.

3.1 Descripción de la arquitectura, fundamentación

La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema. Consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software. Para el desarrollo del componente, se pone en práctica la arquitectura basada en el patrón Modelo Vista Controlador, descrita en el Capítulo 1, esta es una arquitectura definida por tres capas principalmente: la capa de presentación, capa de negocio y capa de acceso a datos.

La capa de presentación está conformada principalmente por páginas XHTML. Estas están compuestas por formularios que mediante controles JSF, Seam UI, Facelets y RichFaces obtienen y validan los datos generados por la interacción del usuario. El uso de estos componentes enriquece el diseño de la interfaz de usuario. Además mediante la utilización de componentes Ajax4Jsf, se logra un efecto más agradable y natural en la interacción con el sistema.

La capa de negocio está constituida por clases controladoras que se encargan de definir la lógica del negocio del componente, así como del manejo y validación de los datos capturados en la capa de presentación. Esta responde a eventos generados por el usuario y es la encargada de generar las peticiones a la capa de modelo cuando se hacen solicitudes de datos, realizando la función de enlace entre las vistas y los modelos.

Por último, la capa de acceso a datos contiene mecanismos para acceder a la información y también para actualizar su estado. Este proceso se logra gracias al uso de componentes Hibernate por los que se encuentra constituida dicha capa. Estos componentes logran abstraer al desarrollador, del gestor de base de datos utilizado, mediante el mapeo de tablas. Lo anterior da la posibilidad de llevar las consultas a un lenguaje de objetos.

3.2 Modelo de diseño

El modelo de diseño crea una representación o modelo de software. Proporciona detalles acerca de la estructura de datos, las arquitecturas, las interfaces y los componentes del software que son necesarios para implementar el sistema.

3.2.1 Patrones de diseño

Los patrones de diseño deben usarse durante el diseño del software. Una vez que se ha desarrollado el modelo de análisis, el diseñador puede examinar una representación detallada del problema que debe resolver y las restricciones impuestas.

GRASP: Patrones de Software para la Asignación General de Responsabilidad. Estos patrones son considerados como una serie de buenas prácticas que deben ser aplicadas en el diseño de software. Para la definición del diseño del componente de notificaciones se utilizaron varios patrones GRASP, los cuales son:

Patrón experto: principio básico en la asignación de responsabilidades, indica en todo momento las clases que cuentan con la información necesaria para poder crear objetos.

Patrón creador: identifica quién debe ser el responsable de la creación o instanciación de nuevos objetos o clases.

Patrón controlador: sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control. Este patrón sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa. La clase intermediaria es la que recibe los datos del usuario y los envía a las distintas clases según el método llamado.

Patrón alta cohesión: indica que la información contenida en las clases debe ser coherente y en la medida de lo posible debe estar relacionada con las mismas.

Patrón bajo acoplamiento: propone que las clases estén lo menos ligadas entre sí que se pueda, disminuyendo la dependencia entre las clases y potenciando la reutilización.

3.2.2 Diagrama de clases del diseño

Muestra definiciones de componentes de software o entidades que son parte de la solución. Incluye las clases, sus atributos, métodos, asociaciones y dependencias.

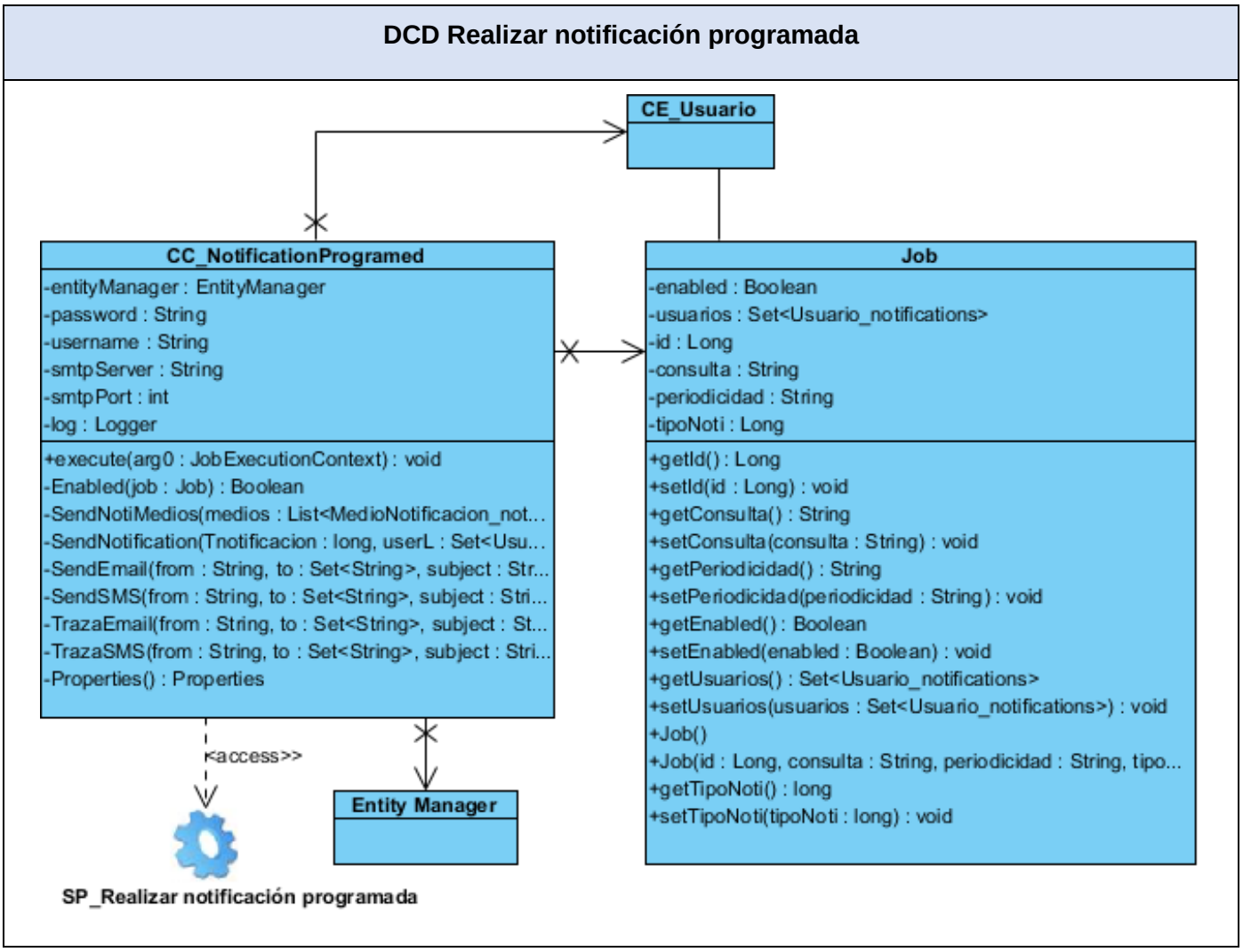

Figura 3. Diagrama de clases del diseño: Realizar notificación programada

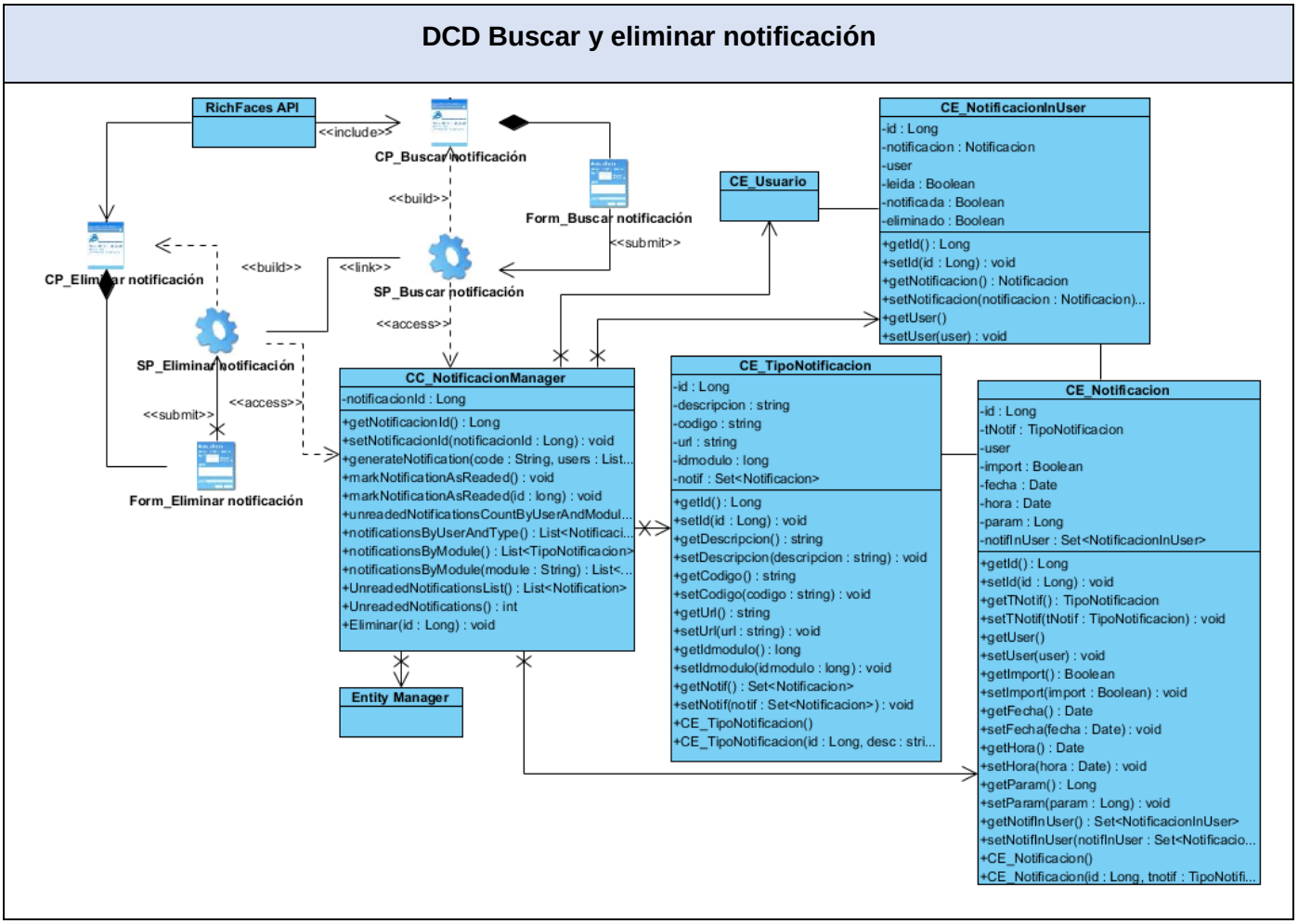

Figura 4. Diagrama de clases del diseño: Buscar y eliminar notificación

DCD Ver datos de notificación y notificación actual

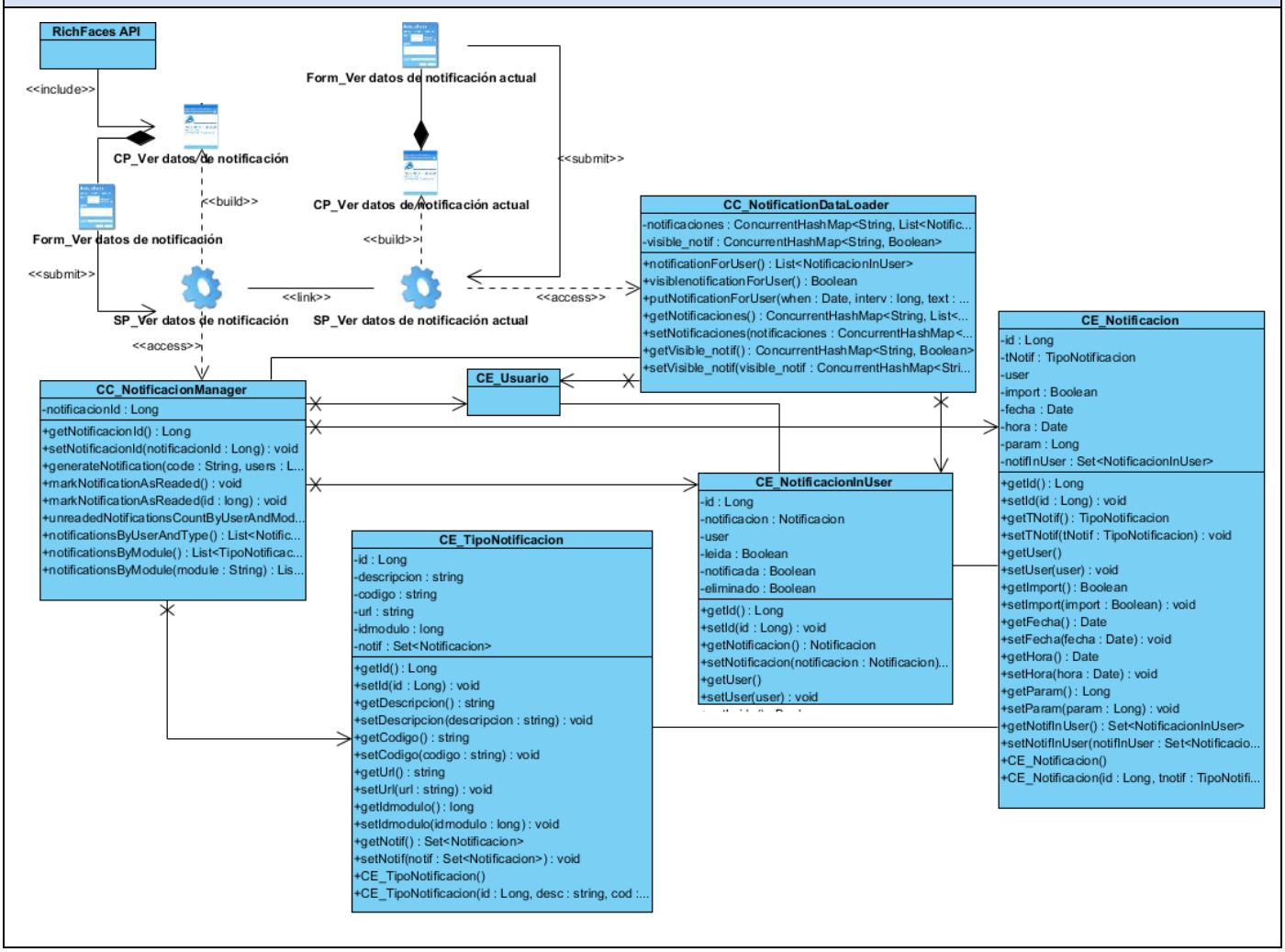

Figura 5. Diagrama de clases del diseño: Ver datos de notificación y notificación actual

Las clases del diseño están agrupadas en:

Páginas servidoras: Están compuestas por componentes Facelets, RichFaces, JSF, Seam UI, así como código HTML. Todo este código será ejecutado en el servidor web, generando páginas clientes que pueden ser representadas por los navegadores web.

Páginas clientes: Están compuestas por código HTML, CSS, JavaScript. Son interpretadas por los navegadores web, presentándole al usuario la interfaz con la que puede interactuar con el sistema.

Formularios: Un formulario HTML es una sección de un documento enmarcado entre etiquetas <form> y que puede contener elementos especiales llamados controles. Los usuarios normalmente "completan" un formulario modificando sus controles (introduciendo texto, seleccionando objetos de un menú, entre otras opciones), envían la información al servidor donde estos son procesados, lo que constituye una manera de obtener en el servidor información entrada por el usuario en el cliente.

Clases controladoras: Las clases controladoras se encargan de la implementación de un caso de uso o un proceso en dependencia de la complejidad de los mismos.

La estructura general del componente estará compuesta por páginas servidoras que construyen páginas clientes, las cuales a su vez contienen formularios que mostrarán y capturarán toda la información enviándola a las páginas servidoras. Estas últimas invocan métodos o responsabilidades en la clase controladora que según la acción solicitada puede modificar o consultar las entidades.

3.2.3 Descripción de las clases y sus atributos

A continuación se realiza la descripción de dos de las clases principales del componente de notificaciones. Estas han sido identificadas en el diseño para su futura implementación, con el objetivo de lograr una comprensión más amplia del componente en cuestión.

Nombre: NotificationManager	
Tipo de clase: Controladora	
Atributo	**Tipo**
notificationId	Long
Para cada responsabilidad	
Nombre:	markNotificationAsReaded() : void

Responsabilidad:	Permite marcar la notificación como leída.
Nombre:	markNotificationAsReaded(notificationId : Long) : void
Responsabilidad:	Permite marcar una notificación obtenida a partir de su id como leída.
Nombre:	unreadedNotificationsCountByUserAndModule() : Long
Responsabilidad:	Permite contar las notificaciones del usuario activo que no han sido leídas.
Nombre:	notificationsByUserAndType(notificationCode : String) : List<NotificacionInUser>
Responsabilidad:	Permite devolver una lista de notificaciones de un tipo específico pertenecientes al usuario activo.
Nombre:	eliminar(id : Long) : void
Responsabilidad:	Permite eliminar una notificación dado su id.
Nombre:	notificationsByModule() : List<TipoNotificacion>
Responsabilidad:	Permite devolver una lista de los tipos de notificaciones recibidos por el módulo activo.
Nombre:	notificationsByModule(nombre : String) : List<TipoNotificacion>
Responsabilidad:	Permite devolver una lista de los tipos de notificaciones recibidos por un módulo especificado por su nombre.
Nombre:	generateNotification(code : String, users : List<String>, roles : List<String>, important : Boolean, param : Long) : void

Responsabilidad:	Permite generar una notificación dado su código, su importancia el id del objeto a notificar y la lista de los destinatarios.
Nombre:	generateNotification(code : String, users : List<Usuario_notifications>, important : Boolean, param : Long) : void
Responsabilidad:	Permite generar una notificación dado su código, su importancia el id del objeto a notificar y la lista de los destinatarios.
Nombre:	UnreadedNotificationsList() : List<Notificacion>
Responsabilidad:	Permite devolver una lista con las notificaciones sin leer del usuario activo.
Nombre:	UnreadedNotifications() : int
Responsabilidad:	Permite devolver la cantidad de notificaciones sin leer del usuario activo.

Tabla 9. Descripción de la clase controladora NotificationManager

Nombre: NotificationDataLoader	
Tipo de clase: Controladora	
Atributo	**Tipo**
notificaciones	ConcurrentHashMap<String, List<NotificacionInUser>>
visible_notificaciones	ConcurrentHashMap<String, Boolean>
Para cada responsabilidad	
Nombre:	notificationForUser() : List<NotificacionInUser>

Responsabilidad:	Devuelve una lista con todas las notificaciones pertenecientes al usuario activo.
Nombre:	visibleNotificationForUser() : Boolean
Responsabilidad:	Permite verificar si la notificación actual es mostrada al usuario.
Nombre:	putNotificationForUser(userName : String, notification : NotificacionInUser) : void
Responsabilidad:	Permite agregar una notificación ya creada a un usuario específico.
Nombre:	loadNewNotifications(when : Date, interval : Long, text : String) : void
Responsabilidad:	Permite cargar las notificaciones nuevas en el atributo notificaciones de la presente clase.

Tabla 10. Descripción de la clase controladora NotificationDataLoader

Nombre: NotificationPrograme d	
Tipo de clase: Controladora	
Atributo	**Tipo**
entityManager	EntityManager
password	String
username	String
smtpServer	String
smtpPort	int

log	Logger
Para cada responsabilidad	
Nombre:	execute() : void
Responsabilidad:	Ejecuta la tarea programada.
Nombre:	Enabled(job : Job) : Boolean
Responsabilidad:	Verifica si debe ejecutar o no la tarea cargada de la base de datos.
Nombre:	SendNotiMedios(medios : List<MedioNotificacion_not>, userL : List<Usuario_not>, tipoNoti : TipoNotificacion_not, objId : Long) : void
Responsabilidad:	Permite enviar las notificaciones por los diferentes medios que le son asignados.
Nombre:	SendNotification(Tnotificacion : long, userL : List<String>, important : Boolean, param : long) : void
Responsabilidad:	Permite construir una notificación y guardarla en la base de datos asociada a los usuarios correspondientes.
Nombre:	SendEmail(from : String, to : List<String>, subject : String, text : String) : void
Responsabilidad:	Permite enviar una notificación por correo electrónico o como mensaje de texto a beeper a los usuarios correspondientes.
Nombre:	TrazaEmail(from : String, to : List<String>, subject : String, text : String) : void
Responsabilidad:	Permite registrar una traza por cada correo electrónico enviado.

Nombre:	SendSMS(from : String, to : List<String>, subject : String, text : String) : void
Responsabilidad:	Permite enviar una notificación por SMS a los usuarios correspondientes.
Nombre:	TrazaSMS(from : String, to : List<String>, subject : String, text : String) : void
Responsabilidad:	Permite registrar una traza por cada SMS enviado.
Nombre:	Properties() : Properties
Responsabilidad:	Permite cargar de la base de datos las Properties necesarias para el envío de notificaciones por correo electrónico y beeper.

Tabla 11. Descripción de la clase controladora NotificationPrograming

Con el desarrollo de este capítulo se describió la arquitectura Modelo-Vista-Controlador que es la utilizada para el desarrollo del HIS del CESIM y se identificaron los patrones GRASP que permitieron adquirir buenas prácticas, por ejemplo, el diseño de las clases se orientó hacia la alta cohesión y el bajo acoplamiento deseados para el componente. Además se crearon los diagramas de clases del diseño identificando las clases que intervienen en cada una de las funcionalidades. Por último se describieron las clases a implementar necesarias para el correcto funcionamiento del componente de notificaciones, así como sus atributos y métodos.

CAPÍTULO 4 IMPLEMENTACIÓN

En este capítulo se describen las clases y subsistemas utilizados en el desarrollo del componente de notificaciones. Se detallan el modelo de datos y los diagramas de despliegue y de componentes. También se muestran los aspectos referentes a la seguridad del componente, las estrategias de codificación y de tratamiento de errores. Por último se implementaran las clases y objetos en ficheros fuente, binarios y ejecutables, entre otros más.

4.1 Modelo de datos

En el modelo de datos se representan gráficamente las entidades con sus atributos y relaciones. Este es usado para definir el mapeo entre las clases del diseño y las estructuras de datos. En este las entidades son objetos de los que se necesita guardar información y los atributos sus características asociadas. Los atributos son clasificados en obligatorios, opcionales, llaves foráneas y llaves primarias. Por último las relaciones definen la asociación entre dos clases. Estas son representadas por una línea que une las dos clases implicadas y además manifiestan la cardinalidad y obligatoriedad de la relación. La descripción de estos elementos proporciona un mejor entendimiento del diagrama que muestra a continuación.

Modelo de datos

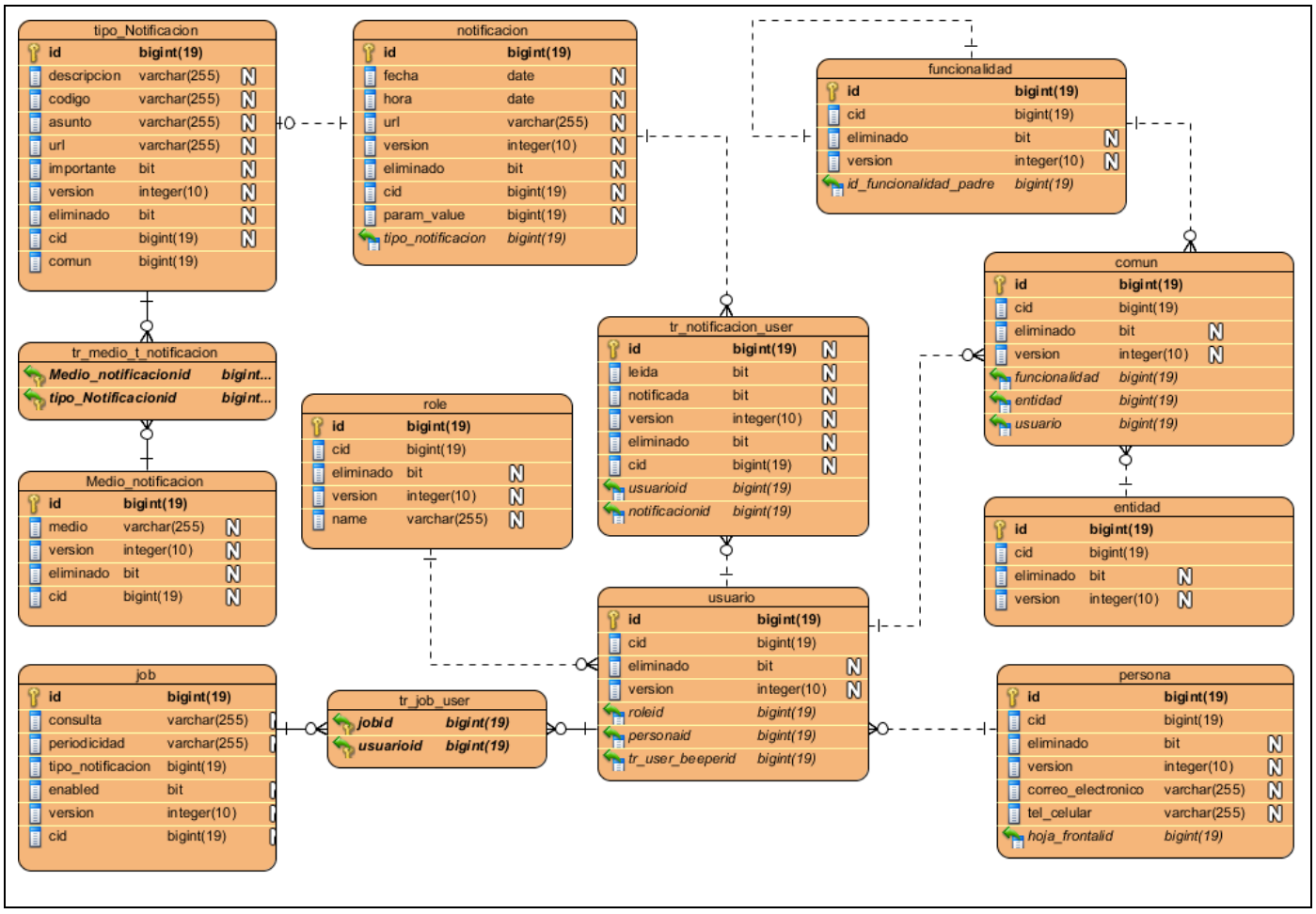

Figura 6. Modelo de datos

4.1.1 Descripción de las tablas de la base de datos

A continuación se describen algunas entidades que forman parte del modelo de datos del componente de notificaciones 2.0. Primeramente se figuran los atributos comunes en todas las entidades y después algunas de las entidades que se destacan.

Atributo	Tipo	Descripción
id	long	Identificador necesario en cada entidad para las referencias en las relaciones entre tablas.

eliminado	boolean	Campo que se pone en verdadero (true) si el campo es eliminado. En caso contrario se mantiene en falso (false).
version	integer	Indica con qué versión de la entidad se está trabajando. Es usado para garantizar que se está trabajando con la versión de la entidad más actualizada que existe en la base de datos.
cid	long	Permite identificar quién realiza alguna acción sobre la entidad.

Tabla 12. Descripción de los atributos comunes entre todas las entidades

comun		
La tabla contiene datos que son necesarios para poder manejar el componente de notificaciones.		
Atributo	**Tipo**	**Descripción**
usuario	long	Llave foránea que hace referencia a la tabla usuario. Se guardará el id del usuario encargado de generar las notificaciones.
entidad	long	Llave foránea que hace referencia a la tabla entidad. Se guardará el id de la entidad encargada de generar las notificaciones.
funcionalidad	long	Llave foránea que hace referencia a la tabla funcionalidad. Se guardará el id del módulo encargado de generar las notificaciones.

Tabla 13. Descripción de la tabla: comun

notificacion		
La tabla contiene los datos de una notificación como son su tipo, la fecha y hora en que fue enviada, el id del objeto sobre el cual se generó la notificación y si es importante o no.		
Atributo	**Tipo**	**Descripción**
id_tipo_notificacion	long	Llave foránea que hace referencia a la tabla tipo_notificacion. No puede ser nula. Se guardara el id del tipo de notificación.
fecha	Date	Fecha en que fue enviada la notificación
hora	Date	Hora en que fue enviada la notificación
param_value	long	Id del objeto sobre el cual fue generado la notificación

Tabla 14. Descripción de la tabla: notificacion

tr_notificacion_user		
La tabla asocia las notificaciones enviadas con los usuarios correspondientes. Además incluye datos sobre si el usuario eliminó o leyó la notificación.		
Atributo	**Tipo**	**Descripción**
id_user	long	Llave foránea que hace referencia a la tabla usuario. Se guardara el id del usuario notificado.
id_notificacion	long	Llave foránea que hace referencia a la tabla notificación. Se guardara el id de la notificación enviada.
leida	boolean	Campo que se pone en verdadero (true) si la notificación es leída por el usuario. En caso contrario se mantiene en falso (false).

notificada	boolean	Campo que se pone en verdadero (true) si la notificación es mostrada al usuario. En caso contrario se mantiene en falso (false).

Tabla 15. Descripción de la tabla: tr_notificacion_user

tipo_notificacion		
La tabla contiene los diferentes tipos de notificaciones, donde define el título de la notificación, su contenido y la dirección del sistema en la cual será mostrada.		
Atributo	**Tipo**	**Descripción**
descripcion	varchar	Campo que guardará el texto de un tipo de notificación.
asunto	varchar	Campo que guardará el asunto que será utilizado en las notificaciones internas y en las enviadas por correo electrónico.
codigo	varchar	Campo que guardará el código de un tipo de notificación, el cual será único para cada tipo de notificación y será el título usado en las notificaciones.
importante	boolean	Campo que se pone en verdadero (true) si la notificación tiene un carácter importante. En caso contrario es falso (false).
url	varchar	Campo que guardará la dirección web dentro del sistema en la cual será mostrada la notificación.
comun	long	Llave foránea que hace referencia a la tabla comun. Se guardará el id de la tabla común que hace referencia a los encargados de generar la notificación.

Tabla 16. Descripción de la tabla: tipo_notificacion

medio_notificacion		
La tabla contiene los diferentes medios por los cuales serán enviadas las notificaciones.		
Atributo	**Tipo**	**Descripción**
medio	varchar	Campo donde se guardará el medio por el cual será enviada la notificación. Ejemplo: Email

Tabla 17. Descripción de la tabla: medio_notificacion

4.2 Modelo de implementación

El modelo de implementación describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos, es decir, toma el resultado del modelo del diseño para generar el código final. Este artefacto describe cómo se implementan los componentes, congregándolos en subsistemas organizados en capas y jerarquías, señalando además las dependencias entre estos. Debido a esto nos permite obtener una visión general de lo que tiene que ser implementado.

4.2.1 Diagrama de componentes

Se utilizan para modelar la vista estática de un sistema. Muestra la organización y las dependencias entre un conjunto de componentes. En él situaremos las librerías, tablas, archivos, ejecutables y documentos que formen parte del sistema. (29)

A continuación se presenta el diagrama de componentes implementado con tres componentes esenciales: Modelo, Vista y Controlador, donde se describen de forma detallada los componentes que forman parte de estos, como son: código fuente, librerías, binarios y ejecutables.

Diagrama de componentes

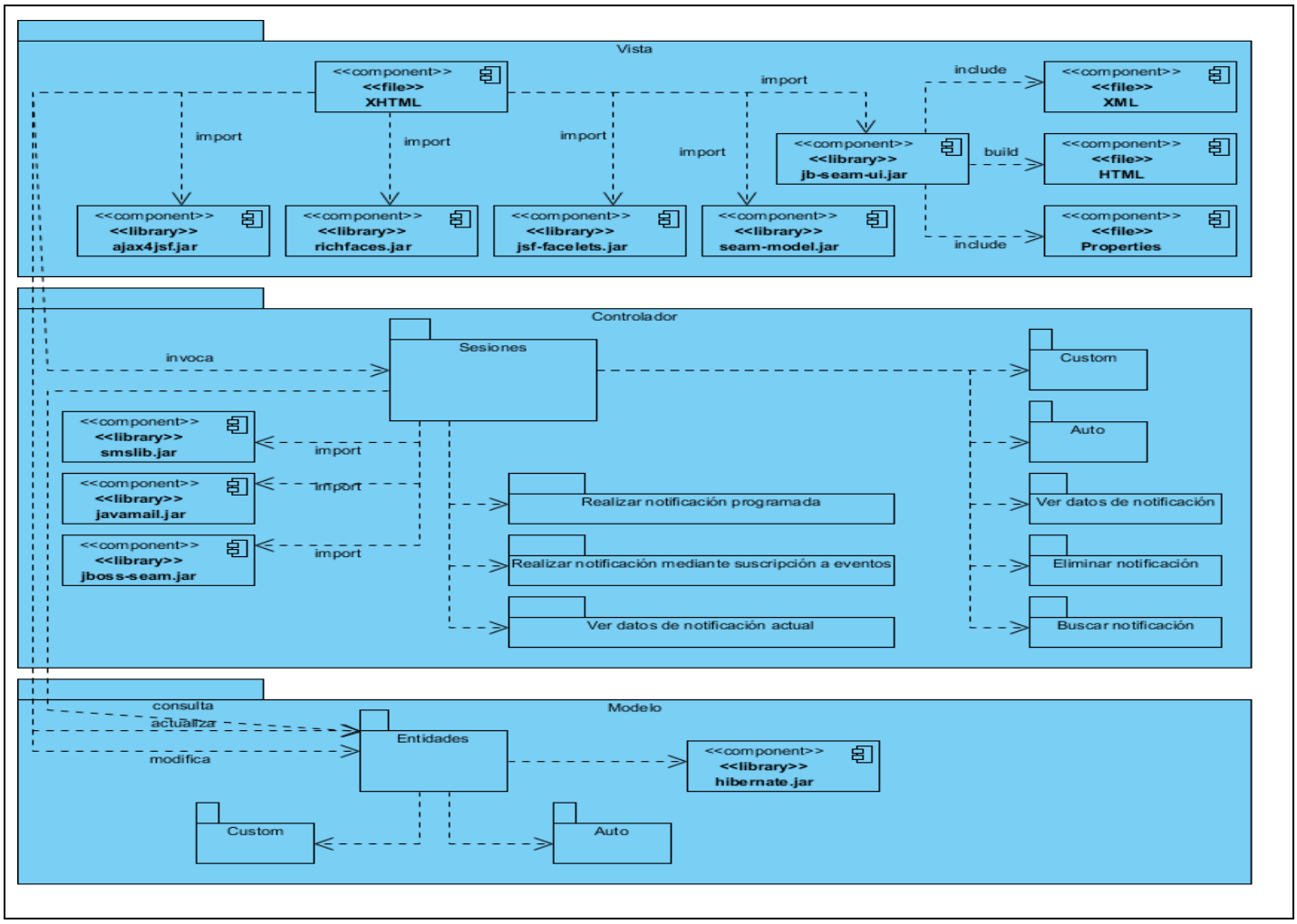

Figura 7. Diagrama de componentes

4.2.2 Diagrama de despliegue

En el diagrama de despliegue se indica la situación física de los componentes lógicos desarrollados. Es decir se sitúa el software en el hardware que lo contiene. (30)

Teniendo en cuenta las características del sistema, el Diagrama de Despliegue quedó estructurado de la siguiente manera:

Diagrama de despliegue

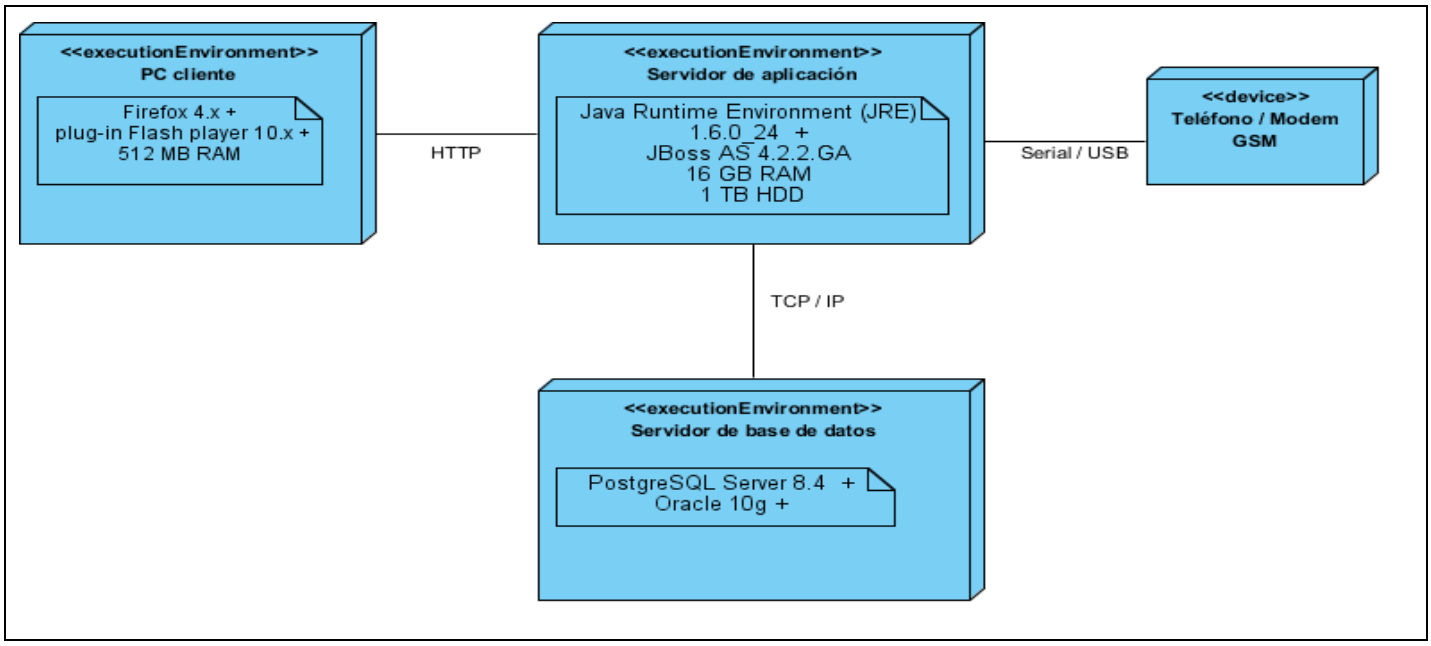

Figura 7. Diagrama de despliegue

Para la implantación y utilización del sistema en un hospital, el usuario debe conectarse al mismo mediante una computadora cliente, haciendo uso de un navegador web. Las peticiones por el protocolo HTTPS serán procesadas por el servidor de aplicaciones, el cual enviará la respuesta al cliente. El servidor de aplicaciones emitirá peticiones por el protocolo TCP / IP (Transfer Control Protocol / Internet Protocol) hacia el servidor de base de datos y se encontrará conectado a un teléfono o módem GSM mediante el puerto Serial / USB.

4.3 Tratamiento de errores

El tratamiento de errores es uno de los aspectos más importantes a tener en cuenta durante el desarrollo del sistema. Toda porción de código donde puedan surgir situaciones inesperadas es tratada mediante el control de excepciones, principalmente donde se ejecutan sentencias que manipulan datos que viajan desde y hacia la base de datos.

4.4 Seguridad

Las notificaciones enviadas a pacientes desde el componente, respetan la confidencialidad de la información que se comunica, por ejemplo, no se emiten resultados de pruebas sanitarias a los pacientes. Solo se les hace saber la disponibilidad de dichos resultados.

En aras de garantizar una correcta protección de los datos manejados, se propone un control de acceso a nivel de usuario y contraseña, con el objetivo de obtener el principio de mínimo privilegio, lo que garantiza el acceso de cada usuario únicamente a los lugares donde tiene permisos. Además el registro de trazas sobre los envíos realizados externos al sistema es vital para mantener un control sobre los mismos. Los datos registrados sólo podrán ser vistos por un administrador del sistema.

4.5 Estrategias de codificación. Estándares y estilos a utilizar

Un estándar de codificación comprende todos los aspectos de la generación de código, de tal manera que sea práctico y entendible para todos los programadores. Por lo general, incluye pautas sobre cómo nombrar las variables y constantes, dónde ubicar los comentarios, así como el uso de paréntesis, guiones y otros elementos presentes en la programación. Éste no detecta los errores existentes, más bien evita la ocurrencia de estos, ya que posibilita un código más legible y en pauta.

4.5.1 Identación

El identado debe ser de dos espacios por bloque de código. No se debe usar el tabulador; ya que este puede variar según la computadora o la configuración de dicha tecla. Los inicios ({) y cierre (}) de ámbito deben estar alineados debajo de la declaración a la que pertenecen y deben evitarse si hay sólo una instrucción.

Para el inicio y fin de bloque se recomienda dejar dos espacios en blanco desde la instrucción anterior para el inicio y fin de bloque {}. Lo mismo sucede para el caso de las instrucciones: if, else, for, while, do while, switch, foreach.

4.5.2 Variables y constantes

El nombre empleado para las variables y constantes, debe permitir que con sólo leerlo se conozca el propósito de la misma. Debe comenzar con la primera letra en minúscula e identificará el tipo de datos al que se refiere. En caso de que sea un nombre compuesto, la segunda palabra, comenzará con letra inicial mayúscula. Ejemplo: envioCorreo. Por último las constantes deben declararse con todas sus letras en mayúsculas.

4.5.3 Comentarios, líneas y espacios en blancos

Comentarios: Se recomienda comentar al inicio de cada clase o función de forma que se especifique el objetivo de la misma así como los parámetros que usa (declarar tipos de datos y objetivo del parámetro), entre otras cosas.

Líneas en blanco: Se recomienda dejar una línea en blanco antes y después de la declaración de una clase o de una estructura y de la implementación de una función.

Espacios en blanco: Se recomienda usar espacios en blanco entre operadores lógicos y aritméticos para lograr una mayor legibilidad del código. Ejemplo: usuario == “root”. No se debe usar espacio en blanco después del corchete abierto y antes del cerrado de un arreglo, luego del paréntesis abierto y antes del cerrado o antes de un punto y coma.

4.5.4 Clases y objetos

El nombre empleado para las clases, objetos, atributos y funciones debe permitir que con sólo leerlo se conozca el propósito de los mismos. Los nombres de las clases deben comenzar con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación PascalCasing. Ejemplo: ProJob. Para el caso de las instancias se comenzará con un prefijo que identificará el tipo de dato, este se escribirá en minúscula.

El nombre que se le da a los atributos de las clases debe comenzar con la primera letra en minúscula y estará en correspondencia al tipo de dato al que se refiere, en caso de que sea un nombre compuesto, la segunda palabra comenzará con mayúscula. Para nombrar las funciones se debe tratar de utilizar verbos que denoten la acción que hacen las mismas. Ejemplo: SendNotificacion(). Si son funciones que obtienen un dato se emplea el prefijo “get” y si fijan algún valor se emplea el prefijo “set”.

Con el desarrollo de este capítulo se pudo dar solución a los requisitos funcionales y no funcionales especificados anteriormente. Para ello se estableció el modelo de datos del sistema, el cual definió las estructuras de datos de las entidades y sus relaciones. Con la realización del diagrama de componentes se definió la estructura en carpetas que presenta el sistema desarrollado, la cual se corresponde con la definida para los componentes del HIS. El diagrama de despliegue brindó los elementos necesarios para una correcta implantación de la aplicación en las entidades que la utilicen. Además se realizó una descripción de las tablas de la base de datos que permitirá un mejor entendimiento a los futuros desarrolladores sobre la información persistida. Durante el proceso de codificación se cumplió con los

estándares y estilos definidos, lo que permitió obtener una aplicación entendible para todos los programadores y fácil de mantener en el transcurso del tiempo.

CONCLUSIONES

Con el desarrollo del componente de notificaciones 2.0 se arribaron a las siguientes conclusiones:

1. El estudio de los sistemas actuales para el envío de notificaciones permitió identificar sistemas con características comunes y algunas de las necesarias en el componente de notificaciones 2.0. Pero estos sistemas no son integrables al HIS del CESIM por ser privativos o estar desarrollados con otras tecnologías.
2. Con el uso de las tecnologías, arquitectura y diseño definidos por el departamento de Sistemas de Gestión Hospitalaria se garantizó el desarrollo de un componente de notificaciones integrable al HIS del CESIM.
3. Los artefactos obtenidos definieron la documentación necesaria de la investigación y sirvieron como guía al desarrollador para la implementación del componente. Además la definición de los requisitos funcionales posibilitó la descripción de las funcionalidades que el sistema debe garantizar, guiando la descripción de los casos de uso.
4. Se desarrollaron funcionalidades que permitieron la integración de los componentes de notificaciones internas y externas, obteniendo un componente único y logrando un menor acoplamiento entre los módulos del HIS del CESIM.

RECOMENDACIONES

Para una posterior versión se recomienda:

1. Implementar una bandeja de salida para las notificaciones externas, para casos de fallos de envío en el momento en que son generadas.
2. Implementar funcionalidades que permitan la configuración general y personalizada del componente de notificaciones de forma gráfica.

REFERENCIAS BIBLIOGRÁFICAS

1. **Real Academia Española.** [En línea] [Citado el: 19 de Febrero de 2014.] http://lema.rae.es/drae/srv/search?key=informática.

2. **TICs.** Tecnología de la informática y las comunicaciones. *Definicion de TIC.* [En línea] [Citado el: 17 de Febrero de 2014.] http://www.tics.org.ar/home/index.php/noticias-destacadas-2/157-definicion-de-tics.

3. **Zayas, Dr. Cs. Carlos Alvarez de.** *METODOLOGIA DE LA INVESTIGACION CIENTIFICA.* Santiago de Cuba : CENTRO DE ESTUDIOS DE EDUCACION SUPERIOR MANUEL F. GRAN, 1995.

4. **Definicion.de.** Definicion.de. *Definición de notificacion .* [En línea] [Citado el: 12 de Diciembre de 2013.] http://definicion.de/notificacion/.

5. **Definicion.de.** Definicion.de. *Definición de evento.* [En línea] [Citado el: 2 de Abril de 2014.] http://definicion.de/evento/.

6. **AESA.** Seguridad Aérea. *Sistema de Notificación de Sucesos.* [En línea] [Citado el: 20 de Febrero de 2014.] http://www.seguridadaerea.gob.es/lang_castellano/g_r_seguridad/notificacion_sucesos/default.aspx.

7. **CRISP.** crisphealth.org. *Encounter-Notification-System-ENS.* [En línea] Agosto de 2013. [Citado el: 07 de Marzo de 2014.] http://crisphealth.org/CRISP-HIE-SERVICES/Encounter-Notification-System-ENS.

8. **Orion Health.** Orion Health. *Automated Alerts.* [En línea] [Citado el: 07 de Marzo de 2014.] http://www.orionhealth.com/glossary-of-terms/automated-alerts.

9. **HIT Intelligent Communications for Healthcare.** Health IT Services. *Notifi Critical Event Notifications.* [En línea] [Citado el: 07 de Marzo de 2014.] http://www.healthitservices.com/solutions/critical-event-notifications/.

10. **Instituto Tecnológico de Colimas.** Instituto Tecnológico de Colimas. *Arquitectura cliente-servidor.* [En línea] [Citado el: 24 de Febrero de 2014.] http://labredes.itcolima.edu.mx/fundamentosbd/sd_u1_6.htm.

11. **Sánchez González, Carlos.** Proyecto de fin de carrera. *Aplicaciones en capas.* [En línea] 28 de Septiembre de 2004. [Citado el: 24 de Febrero de 2014.] http://oness.sourceforge.net/proyecto/html/ch03s02.html.

12. **Universidad de Valladolid.** Departamento de informática Universidad de Valladolid. *Java.* [En línea] [Citado el: 23 de Febrero de 2014.] http://www.infor.uva.es/~jmrr/tgp/java/JAVA.html.

13. **Oracle.** Oracle.com. *Java.* [En línea] [Citado el: 23 de Febrero de 2014.] http://www.oracle.com/us/technologies/java/overview/index.html.

14. **Oracle.** Oracle.com *JavaServer Faces Technologies.* [En línea] [Citado el: 25 de Febrero de 2014.] http://www.oracle.com/technetwork/java/javaee/overview-140548.html.

15. **JBOSS.** JBoss.org. *RichFaces.* [En línea] [Citado el: 24 de Febrero de 2014.] http://www.jboss.org/richfaces.

16. **JavaBeat.** JavaBeat. *Introduction to Ajax4Jsf.* [En línea] [Citado el: 25 de Febrero de 2014.] http://www.javabeat.net/introduction-to-ajax4jsf/.

17. **Oracle.** oracle.com. *What is Facelets.* [En línea] [Citado el: 25 de Febrero de 2014.] http://docs.oracle.com/javaee/6/tutorial/doc/gijtu.html.

18. **W3C.** W3C. *XHTML.* [En línea] [Citado el: 25 de Febrero de 2014.] http://www.w3c.es/Divulgacion/GuiasBreves/XHTML.

19. **King Gavin, Muir Pete, Richards Norman, Bryzak Shane, Yuan Michael, Youngstrom Mike, Bauer Christian, Balunas Jay, Allen Dan, Rydahl Andersen Max, Bernard Emmanuel, Karlsson Nicklas, Roth Daniel, Drees Matt, Orshalick Jacob, and Novotny Marek.** SeamFramework.org. *Seam - Contextual Components.* [En línea] [Citado el: 25 de Febrero de 2014.] http://docs.jboss.org/seam/snapshot/en-US/html/index.html.

20. **Hibernate.org.** Hibernate.org. *Hibernate Reference Documentation.* [En línea] 30 de Junio de 2009. [Citado el: 25 de Febrero de 2014.] http://docs.jboss.org/hibernate/core/3.2/reference/en/pdf/.

21. **JBOSS.** Jboss.org. *EJB3.* [En línea] [Citado el: 25 de Febrero de 2014.] http://www.jboss.org/ejb3.

22. **Sun Microsystems.** Java en Castellano. *Catálogo de Patrones de Diseño J2EE. I. Capa de Presentación.* [En línea] [Citado el: 25 de Febrero de 2014.] http://www.programacion.net/java/tutorial/patrones/1/.

23. **Jamae David, Johnson Peter.** *JBoss in Action.* 2009. 1933988029.

24. **Universidad Autonoma de Baja California.** Curso de Análisis Orientado a Objetos. *El Proceso Unificado de Desarrollo de Software.* [En línea] 2004. [Citado el: 2 de Marzo de 2014.] http://yaqui.mxl.uabc.mx/~molguin/as/RUP.htm.

25. **OMG.** uml.org. *Introduction to OMG's UML.* [En línea] Julio de 2005. [Citado el: 2 de Marzo de 2014.] http://www.omg.org/gettingstarted/what_is_uml.htm.

26. **JBOSS.** JBoss Community. *JBoss Developer Studio.* [En línea] [Citado el: 2 de Marzo de 2014.] https://www.jboss.org/products/jbds.

27. **The PostgreSQL Global Development Group.** PostgreSQL. *About.* [En línea] [Citado el: 2 de Marzo de 2014.] http://www.postgresql.org/about/.

28. **Visual Paradigm.** Visual Paradigm. *Visual Paradigm for UML.* [En línea] [Citado el: 2 de Marzo de 2014.] http://www.visual-paradigm.com/product/vpuml/.

29. **Programación en castellano.** Programacion en castellano. *Introducción a UML.* [En línea] 2013. [Citado el: 20 de Mayo de 2014.] http://www.programacion.com/articulo/introduccion_a_uml_181/6.

30. **Programación en castellano.** Programación en castellano. *Introducción a UML.* [En línea] 2013. [Citado el: 20 de Mayo de 2014.] http://www.programacion.com/articulo/introduccion_a_uml_181/7.

31. *Definicion de TIC.* [En línea] [Citado el: 19 de Febrero de 2014.] http://www.tics.org.ar/home/index.php/noticias-destacadas-2/157-definicion-de-tics.

32. **Definicion.de.** Definicion.de. *Definición de alerta.* [En línea] [Citado el: 18 de Febrero de 2014.] http://definicion.de/alerta/.

33. **Word Reference.** WordReference.com. *Destinatario.* [En línea] [Citado el: 18 de Febrero de 2014.] http://www.wordreference.com/definicion/destinatario.

34. **Dictionarist.com.** Dictionarist.com. *Definicion de trace.* [En línea] [Citado el: 22 de Febrero de 2014.] http://definicion.dictionarist.com/trace.

35. **developer.android.com.** developer.android.com. *Notifications.* [En línea] [Citado el: 24 de Febrero de 2014.] http://developer.android.com/guide/topics/ui/notifiers/notifications.html.

36. **EPV/EHU.** Portal de la Universidad del País Vasco. *Gestión Servicios - Notificaciones.* [En línea] [Citado el: 24 de Febrero de 2014.] https://gestion-servicios.ehu.es/notificacion/.

37. **Indah Mohd Amin, Surya Sumarni Hussein, Wan Abdul Rahim Wan Mohd Isa.** International Proceedings of Economics Development and Research. *Assessing User Satisfaction of using HIS in Malaysia.* [En línea] 2011. [Citado el: 2 de Marzo de 2014.] http://www.ipedr.com/vol5/no1/45-H00097.pdf.

38. **Jaspersoft Corporation.** Community Jaspersoft. *iReport Designer Getting Started.* [En línea] [Citado el: 2 de Marzo de 2014.] http://community.jaspersoft.com/wiki/ireport-designer-getting-started.

39. **Jaspersoft Corporation.** Comunidad JasperSoft. *JasperReports Library.* [En línea] [Citado el: 07 de Marzo de 2014.] http://community.jaspersoft.com/project/jasperreports-library.

40. **Yorio, Darío.** *Identificación y clasificación de patrones en el diseño de aplicaciones móviles.* s.l. : Universidad de La Plata.

41. **Universidad Nacional Abierta y a Distancia.** Universidad Nacional Abierta ya Distancia. *Lección 14. Características Avanzadas de las Clases y relaciones.* [En línea] [Citado el: 8 de Abril de 2014.] http://datateca.unad.edu.co/contenidos/200609/exeuml/leccin_14_caractersticas_avanzadas_de_las_clases_y_relaciones.html.

BIBLIOGRAFÍA

- **AESA.** Seguridad Aérea. *Sistema de Notificación de Sucesos.* [En línea] [Citado el: 20 de Febrero de 2014.] http://www.seguridadaerea.gob.es/lang_castellano/g_r_seguridad/notificacion_sucesos/default.aspx.
- **Bali, Michal.** www.packtpub.com. *Drools JBoss Rules 5.0.* [En línea] 2009. [Citado el: 5 de Abril de 2014.] www.packtpub.com/Books.
- **CRISP.** crisphealth.org. *Encounter-Notification-System-ENS.* [En línea] Agosto de 2013. [Citado el: 07 de Marzo de 2014.] http://crisphealth.org/CRISP-HIE-SERVICES/Encounter-Notification-System-ENS.
- **Definicion.de.** Definicion.de. *Definición de alerta.* [En línea] [Citado el: 18 de Febrero de 2014.] http://definicion.de/alerta/.
- **Definicion.de.** Definicion.de. *Definición de evento.* [En línea] [Citado el: 2 de Abril de 2014.] http://definicion.de/evento/.
- **Definicion.de.** Definicion.de. *Definición de notificacion .* [En línea] [Citado el: 12 de Diciembre de 2013.] http://definicion.de/notificacion/.
- ***Definicion de TIC.*** [En línea] [Citado el: 19 de Febrero de 2014.] http://www.tics.org.ar/home/index.php/noticias-destacadas-2/157-definicion-de-tics.
- **developer.android.com.** developer.android.com. *Notifications.* [En línea] [Citado el: 24 de Febrero de 2014.] http://developer.android.com/guide/topics/ui/notifiers/notifications.html.
- **Dictionarist.com.** Dictionarist.com. *Definicion de trace.* [En línea] [Citado el: 22 de Febrero de 2014.] http://definicion.dictionarist.com/trace.
- **EPV/EHU.** Portal de la Universidad del País Vasco. *Gestión Servicios - Notificaciones.* [En línea] [Citado el: 24 de Febrero de 2014.] https://gestion-servicios.ehu.es/notificacion/.

- **Gonzáles Benito, Eneko.** Java Hispano. *Introducción al API Reflection (Reflexión) de Java.* [En línea] 2004. [Citado el: 26 de Febrero de 2014.] www.javahispano.org/storage/contenidos/reflection.pdf.
- **Grau, Ricardo, Correa, Cecilia y Rojas, Mauricio.** *Metodología de la investigación, Segunda edición.* s.l. : Fondo Editorial Coruniversitaria, 2004. 958-8028-10-8.
- **Hibernate.org.** Hibernate.org. *Hibernate Reference Documentation.* [En línea] 30 de Junio de 2009. [Citado el: 25 de Febrero de 2014.] http://docs.jboss.org/hibernate/core/3.2/reference/en/pdf/.
- **HIT Intelligent Communications for Healthcare.** Health IT Services. *Notifi Critical Event Notifications.* [En línea] [Citado el: 07 de Marzo de 2014.] http://www.healthitservices.com/solutions/critical-event-notifications/.
- **IBM.** Java Diagnostics Guide. *how to write a custom loader.* [En línea] [Citado el: 23 de Febrero de 2014.] http://publib.boulder.ibm.com/infocenter/javasdk/v1r4m2/index.jsp?topic=%2Fcom.ibm.java.doc.diagnostics.142%2Fhtml%2Fid1100.html.
- **Indah Mohd Amin, Surya Sumarni Hussein, Wan Abdul Rahim Wan Mohd Isa.** International Proceedings of Economics Development and Research. *Assessing User Satisfaction of using HIS in Malaysia.* [En línea] 2011. [Citado el: 2 de Marzo de 2014.] http://www.ipedr.com/vol5/no1/45-H00097.pdf.
- **Instituto Tecnológico de Colimas.** Instituto Tecnológico de Colimas. *Arquitectura cliente-servidor.* [En línea] [Citado el: 24 de Febrero de 2014.] http://labredes.itcolima.edu.mx/fundamentosbd/sd_u1_6.htm.
- **Jamae David, Johnson Peter.** *JBoss in Action.* 2009. 1933988029.
- **JasperSoft Corporation.** Comunidad JasperSoft. *JasperReports Library.* [En línea] [Citado el: 07 de Marzo de 2014.] http://community.jaspersoft.com/project/jasperreports-library.
- **Jaspersoft Corporation .** Community Jaspersoft. *iReport Designer Getting Started.* [En línea] [Citado el: 2 de Marzo de 2014.] http://community.jaspersoft.com/wiki/ireport-designer-getting-started.

- **JavaBeat.** JavaBeat. *Introduction to Ajax4Jsf.* [En línea] [Citado el: 25 de Febrero de 2014.] http://www.javabeat.net/introduction-to-ajax4jsf/.
- **JBOSS.** Jboss.org. *EJB3.* [En línea] [Citado el: 25 de Febrero de 2014.] http://www.jboss.org/ejb3.
- **JBOSS.** JBoss Community. *JBoss Developer Studio.* [En línea] [Citado el: 2 de Marzo de 2014.] https://www.jboss.org/products/jbds.
- **JBOSS.** JBoss.org. *RichFaces.* [En línea] [Citado el: 24 de Febrero de 2014.] http://www.jboss.org/richfaces.
- **JBOSS.** seamframework.org. *seam tutorial.* [En línea] [Citado el: 23 de Febrero de 2014.] http://docs.jboss.org/seam/snapshot/en-US/html/tutorial.html.
- **King, Gavin, y otros.** Documentación de referencia de hibernate. *Documentación de referencia de hibernate.* [En línea] [Citado el: 21 de Febrero de 2014.] docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html.
- **King, Gavin, Muir Pete, Richards Norman, Bryzak Shane, Yuan Michael, Youngstrom Mike, Bauer Christian, Balunas Jay, Allen Dan, Rydahl Andersen Max, Bernard Emmanuel, Karlsson Nicklas, Roth Daniel, Drees Matt, Orshalick Jacob, and Novotny Marek.** SeamFramework.org. *Seam - Contextual Components.* [En línea] [Citado el: 25 de Febrero de 2014.] http://docs.jboss.org/seam/snapshot/en-US/html/index.html.
- **OMG.** uml.org. *Introduction to OMG's UML.* [En línea] Julio de 2005. [Citado el: 2 de Marzo de 2014.] http://www.omg.org/gettingstarted/what_is_uml.htm.
- **Oracle.** oracle.com. *Java.* [En línea] [Citado el: 23 de Febrero de 2014.] http://www.oracle.com/us/technologies/java/overview/index.html.
- **Oracle.** oracle.com. *JavaServer Faces Technologies.* [En línea] [Citado el: 25 de Febrero de 2014.] http://www.oracle.com/technetwork/java/javaee/overview-140548.html.
- **Oracle.** oracle.com. *What is Facelets.* [En línea] [Citado el: 25 de Febrero de 2014.] http://docs.oracle.com/javaee/6/tutorial/doc/gijtu.html.

- **Orion Health.** Orion Health. *Automated Alerts.* [En línea] [Citado el: 07 de Marzo de 2014.] http://www.orionhealth.com/glossary-of-terms/automated-alerts.
- **Programacion en castellano.** Programacion en castellano. *Introducción a UML.* [En línea] 2013. [Citado el: 20 de Mayo de 2014.] http://www.programacion.com/articulo/introduccion_a_uml_181/6.
- **Programación en castellano.** Programación en castellano. *Introducción a UML.* [En línea] 2013. [Citado el: 20 de Mayo de 2014.] http://www.programacion.com/articulo/introduccion_a_uml_181/7.
- **Real Academia Española.** Real Academia Española. *Real Academia Española.* [En línea] [Citado el: 19 de Febrero de 2014.] http://lema.rae.es/drae/srv/search?key=informática.
- **Ruwanpathirana, Kalani.** kalani's tech blog. *How to Write a Custom Class Loader to Load Classes from a Jar.* [En línea] [Citado el: 12 de Marzo de 2014.] http://kalanir.blogspot.com/2010/01/how-to-write-custom-class-loader-to.html.
- **Sánchez González, Carlos.** Proyecto de fin de carrera. *Aplicaciones en capas.* [En línea] 28 de Septiembre de 2004. [Citado el: 24 de Febrero de 2014.] http://oness.sourceforge.net/proyecto/html/ch03s02.html.
- **Sun Microsystems.** Java en Castellano. *Catálogo de Patrones de Diseño J2EE. I. Capa de Presentación.* [En línea] [Citado el: 25 de Febrero de 2014.] http://www.programacion.net/java/tutorial/patrones/1/.
- **The PostgreSQL Global Development Group.** PostgreSQL. *About.* [En línea] [Citado el: 2 de Marzo de 2014.] http://www.postgresql.org/about/.
- **TICs.** Tecnología de la informática y las comunicaciones. *Definicion de TIC.* [En línea] [Citado el: 17 de Febrero de 2014.] http://www.tics.org.ar/home/index.php/noticias-destacadas-2/157-definicion-de-tics.
- **Tutorials Point.** Tutorials Point - Simply Easy Learning. *EJB - Interceptores.* [En línea] [Citado el: 11 de Marzo de 2014.] http://www.tutorialspoint.com/ejb/ejb_interceptors.htm.
- **Universidad Autonoma de Baja California.** Curso de Análisis Orientado a Objetos. *El Proceso Unificado de Desarrollo de Software.* [En línea] 2004. [Citado el: 2 de Marzo de 2014.] http://yaqui.mxl.uabc.mx/~molguin/as/RUP.htm.

- **Universidad de Valladolid.** Departamento de informática Universidad de Valladolid. *Java.* [En línea] [Citado el: 23 de Febrero de 2014.] http://www.infor.uva.es/~jmrr/tgp/java/JAVA.html.
- **Universidad Nacional Abierta y a Distancia.** Universidad Nacional Abierta ya Distancia. *Lección 14. Características Avanzadas de las Clases y relaciones.* [En línea] [Citado el: 8 de Abril de 2014.] http://datateca.unad.edu.co/contenidos/200609/exeuml/leccin_14_caractersticas_avanzadas_de_las_clases_y_relaciones.html.
- **Visual Paradigm.** Visual Paradigm. *Visual Paradigm for UML.* [En línea] [Citado el: 2 de Marzo de 2014.] http://www.visual-paradigm.com/product/vpuml/.
- **W3C.** W3C. *XHTML.* [En línea] [Citado el: 25 de Febrero de 2014.] http://www.w3c.es/Divulgacion/GuiasBreves/XHTML.
- **Word Reference.** WordReference.com. *Destinatario.* [En línea] [Citado el: 18 de Febrero de 2014.] http://www.wordreference.com/definicion/destinatario.
- **Yorio, Darío.** *Identificación y clasificación de patrones en el diseño de aplicaciones móviles.* s.l. : Universidad de La Plata.
- **Zayas, Dr. Cs. Carlos Alvarez de.** *METODOLOGIA DE LA INVESTIGACION CIENTIFICA.* Santiago de Cuba : CENTRO DE ESTUDIOS DE EDUCACION SUPERIOR MANUEL F. GRAN, 1995.

ANEXOS

Anexo 1: Entrevista no estructurada realizada en el departamento de Sistemas de Gestión Hospitalaria en Marzo del 2014.

Objetivo: Obtener conocimientos que permitan identificar los requisitos funcionales que debe presentar el componente de notificaciones 2.0.

1. Módulos que utilizan el componente de notificaciones internas.
2. Procesos actuales de notificación (internas y externas).
3. Tipos de notificaciones que existen actualmente.
4. Situaciones que ocurren actualmente en el HIS del CESIM y no son notificadas.

GLOSARIO DE TÉRMINOS

Transaccionalidad: Se refiere a la interacción con una estructura de datos compleja, donde los procesos deben aplicarse uno después del otro de manera similar a una interacción atómica.

Lenguaje multihilos: Lenguaje de programación que permite la ejecución de varias actividades en forma simultánea, tanto en un programa creado en el lenguaje como en la parte interna del propio lenguaje.

Plug-in (plugin): En la informática se refiere a un complemento o una aplicación que se relaciona con otra para aportarle una nueva funcionalidad. Permite que los desarrolladores externos colaboren con la aplicación principal extendiendo sus funciones.